Alexin Zoltán

Szabályalapú gépi tanulási módszerek és alkalmazásaik természetes nyelvi problémák megoldására

Doktori értekezés

Témavezető: *Gyimóthy Tibor, PhD*

Informatika Doktori Program
Szegedi Tudományegyetem
2002

Tartalomjegyzék

1. Bevezetés

A dolgozat a gépi tanuló algoritmusokkal és egyes tanuló algoritmusok gyakorlati alkalmazásaival foglalkozik. Ez a terület a mesterséges intelligencia tárgykörébe tartozó kutatási ág. A tanuló algoritmusok a számítógépek olyan alkalmazását jelentik, amelyben a tanulás képességével rendelkező számítógépes rendszerek módosítani tudják saját működésüket a használat során nyert tapasztalatok alapján. A tanuló algoritmusok tanulmányozása elvezethet az emberi gondolkodás, következtetés jobb megismeréséhez. Az általános célú tanuló algoritmusok egy kezdeti hipotézis valamint tréning példák felhasználásával javított és a példákkal konzisztens hipotéziseket képesek előállítani. A tréning időszakban igényelt jelentős mennyiségű emberi és gépi erőforrás költség várhatóan megtérül amiatt, mert mindezt elegendő a szoftver fejlesztési fázisában egyszer elvégezni. Az alábbiakban három fontos alkalmazási területet külön is megemlítek, ahol rutinszerűen használnak tanuló algoritmusokat. Ezek az adatbányászat, a bioinformatika és a természetesnyelv-feldolgozás.

A tanuló algoritmusok kutatásának egyik mára már lényegében önállósult ága az adatbányászat [Bha99] (*data mining*). Nagy adatbázisokat használnak ma már az üzleti életben, a bankrendszerben, a mérnöki tudományban, a társadalombiztosításban és a természettudományban. A valóban releváns információ kinyerése ezekből az adatbázisokból fontos gyakorlati feladat. A számítógépes tanulással kapcsolatos kutatások az alábbi kérdésre helyezik a hangsúlyt: lehetséges-e egy olyan leíró modell meghatározása a konkrét adatok alapján, amely egyszerre pontos is és az ember számára könnyen érthető. Több ígéretes projekt indult a gépi tanulás alkalmazására pl. diagnózis tanulására egészségügyi adatokból, időjárás előrejelzés légköri mérési adatok alapján, tipikus vásárlói szokások feltérképezése hipermarketek eladási adatbázisai alapján.[1]

Hasonló az alapfeladata a bioinformatikán belül a genetikus számítástechnikának (*genetic computing*). A DNS szekvenálási technológia kifejlesztése óta számos különböző keresési feladat fogalmazódott meg, amelyeket hatalmas DNS szekvencia adatbázisokon kell végrehajtani. A legfontosabb problémák egyike a DNS által kódolt fehérje szerkezetének kutatása (*protein structure engineering*), vírus, baktérium fehérjetöredékek keresése, gyógyszermolekulák tervezése (*drug design, drug engineering, learning structure-activity relation*). A fehérjék alapvetően aminosavakból álló láncok. Az alkotó aminosavak sorrendjét a fehérjék elsődleges szerkezetének nevezik. A fehérjemolekulák térbeli szerkezete az ún. másodlagos szerkezet. A tanuló algoritmusok segítségével megjósolható, hogy egy fehérje egy adott pozíción található aminosava része-e egy jellegzetes 3-dimenziós alakzatnak (α-helix, β-lánc, sávok, egyenes szakaszok). A tanulási példák ismert molekulák adataiból állnak, háttér tudásként pedig használhatók az adott pozíció szomszédságában található aminosavak, valamint az egyes aminosavak fizikai és kémiai tulajdonságai (hidrofóbia, töltés, méret, polaritás).

A tanuló algoritmusok alkalmazásának ugyancsak fontos területe a természetesnyelv-feldolgozás. Egy természetes nyelv milliós nagyságrendben tartalmaz szavakat. Ragozó (agglutináló) nyelvek esetén, mint amilyen a magyar is, a sok rag és jel miatt ez a szám elérheti a 10 milliót. Az írott szöveg megértéséhez a kulcs az összefüggések és a nyelvi szerkezetek elemzése, amely felhasználja a szavak morfo-szintaktikai attribútumait. Az Internet elterjedésével a mindenki által könnyen elérhető írott szöveges információ mennyisége drámai módon növekedett. Ennek a heterogén, soknyelvű anyagnak a bármilyen kis mértékű rendezése, feldolgozása, kivonatolása is nagy fontosságú, ezért került a természetesnyelv-feldolgozás a tudományos érdeklődés középpontjába. A természetesnyelv-feldolgozáshoz szorosan kapcsolódnak a beszédfelismeréssel kapcsolatos kutatások. A tanuló algoritmusok hatékonyan alkalmazhatók például a beszédfelismerésben, a szófaji egyértelműsítésben, a főnévi szerkezetek kijelölésében. A tanulási példák annotált szöveg vagy hang minták. A háttértudást nyelvészeti, fonológiai szabályszerűségek alkotják. A feladat egy ismeretlen beszéd vagy szöveg automatikus annotálása. A 4. Fejezet elején a szabályalapú tanuló algoritmusok még számos további érdekes alkalmazás ismertetésére kerül sor.

[1] UCI (University of California, Irvine) Machine Learning Archive, Knowledge Discovery and Data Mining (Számítógépes tanulás, adatbányászat): http://www.ics.uci.edu/~mlearn/Machine-Learning.html

T. Mitchel [Mit97] az alábbiak szerint határozta meg a tanuló programok fogalmát:

1.1. Definíció: Adottak a következők: Z egy számítógépes program, E tapasztalati tények (tréning adatok) egy halmaza, T elvégzendő (teszt) feladatok egy halmaza és P egy végrehajtási teljesítmény mérték. A Z program **tanul** az E gyakorlati tapasztalatokból, ha a Z programot az E gyakorlati példák feldolgozása után ismételten lefuttatva a T teszt feladatokon, a Z program P mérték szerinti teljesítménye javul.

Tanuló algoritmusok készítésére számos elméleti modell létezik. A modell alkalmas eszközt kell, hogy biztosítson a tanulási (tréning) és a teszt példák számítógépes reprezentációjára, ha van, akkor egy tudásbázisnak és végül magának a Z programnak a reprezentációjára is. A reprezentáció lehet például egy mesterséges neurális hálózat [Bis95], [Hay94] vagy egy Markov hálózat [Man00].

Egy további lehetőség az, hogy a Z programot utasítások (kiszámítási szabályok, képletek vagy formulák) sorozataként reprezentáljuk. A tanulásnak ezt az ágát általában a szimbolikus tanulás néven ismerik. A szimbolikus tanuláson belül különösen fontos terület a szabály-halmaz tanulás. Szabályokon *„ha-akkor"* típusú formulákat kell érteni, amelyekkel definiálni lehet egy új relációt ismert relációk felhasználásával. A szabály-halmazok tanulására szolgáló algoritmusokat nevezzük szabályalapú tanuló algoritmusoknak.

A jelenlegi tanuló algoritmusok hatékony felhasználása egyelőre még komoly szakmai felkészültséget igényel (a kutatások egy része éppen a könnyebb felhasználhatóságra irányul), ettől függetlenül azonban nem lehet lemondani az alkalmazásukról akkor, amikor a megoldandó probléma emberi léptékkel mérve nagyon nagy vagy bonyolult. Fontos mindjárt előrebocsátani, hogy a tanuló programok maguk is *algoritmusok alapján működnek*, azaz a hipotéziseiket valamilyen szabályszerűség szerint *generálják* (egy keresési teret járnak be). A programok a rendelkezésükre álló idő alatt nagy keresési tereket tudnak megvizsgálni, és közben folyamatosan ellenőrzik a hipotéziseiket. Nem képesek azonban arra a kreativitásra, ami továbbra is csak az ember sajátja: Nem rendelkeznek asszociatív képességekkel, nem fognak tudni analógiákat felfedezni két teljesen különböző probléma között, nem tudnak önállóan új reprezentációkat feltalálni.

Az alábbi definíció a gépi tanulás legalapvetőbb mérföldkövét, az ún. induktív tanulási hipotézist ismerteti. Ez a hipotézis mutatja be leginkább a tanuló algoritmusok lehetőségeit és korlátait. Egyrészt a tréning halmaz elemeinek számát folyamatosan tudjuk növelni (a rendelkezésre álló technika által kijelölt határig), ezzel egyre kifinomultabb rendszereket tanulhatunk meg, mindazonáltal a teljes rendszer teljesítménye csak valószínűségi szempontból fog javulni, előre nem látott események esetén a program továbbra is hibázhat. A hipotézis valójában összhangban van az emberi társadalom eddigi fejlődésével, amennyiben a társadalom mai fejlettségi szintjére a véges példa halmazokból történő tanulás módszerével jutott el.

1.2. Definíció: Az induktív tanulási hipotézis [Mit97]: ha egy megtalált hipotézis jól közelíti a megtanulandó formulát (szabályt, predikátumot) egy elegendően nagy példa halmazon, akkor az jól fogja közelíteni a megtanulandó formulát a példák között elő nem fordult esetekben is.

Az induktív tanulási hipotézis teremti meg az alapot a hitelesített tanulási adatbázisok felhasználására. Számos gyakorlati esetben ugyanis a feladat megfogalmazásán túlmenően további elméleti keret nem áll rendelkezésre, az egyetlen forrás egy adatbázis, amelyben tapasztalati adatok találhatók a tanulmányozandó jelenségre vonatkozóan. A tanuló algoritmusok által adott eredmény minőségét az határozza meg leginkább, hogy a felhasznált tréning adatbázis mennyire elfogadott, szabványos és hiteles. Az utóbbi években ilyen adatbázisok készítése elterjedőben van a világban, lassan egy piac is formálódik ezen a területen, ugyanis az adatbázisok nemcsak tréning feladatokra (kutatásra), hanem egy más módon elkészített szoftver rendszer tesztelésére (és ily módon hitelesítésére) is alkalmasak.

A dolgozatban a szerző ismertet egy saját fejlesztésű interaktív tanuló algoritmust, a későbbiekben pedig ennek és más szabályalapú tanuló algoritmusoknak gyakorlati alkalmazása területén kifejtett tudományos eredményeit. A 2. Fejezetben a szabályalapú tanuló algoritmusok néhány jelentősebb képviselője kerül bemutatásra, a 3. Fejezetben a szerző és munkatársai által kifejlesztett IMPUT tanuló algoritmus, a 4. Fejezetben számos természetes nyelvi alkalmazás található, amelyet az összefoglaló követ az 5. Fejezetben.

A szerző tanuló algoritmusok alkalmazásával kapcsolatos tudományos tevékenységét a 3. Fejezet és a 4. Fejezet tartalmazza. A 3. Fejezetben az IMPUT abduktív tanuló algoritmus ismertetésére kerül sor, amely a szerző saját eredménye. Az IMPUT algoritmust a szerző implementálta is, és számos alkalmazás kifejlesztésére is sor került. Az IMPUT rendszer sikeresen alkalmazható volt a beszédfelismerésben magyar magánhangzók felismerésére, illetve EKG hullámok szintaxisának tanulására.

Az IMPUT rendszer a svéd Boström által kifejlesztett SPECTRE algoritmus egy jelentős továbbfejlesztése. Az IMPUT algoritmus a beépített IDTS hibakereső modulnak köszönhetően jelentősen javította a megtanult program minőségét. Bár a megtanult programok egyformán helyesek voltak mindkét tanuló program esetében, az IMPUT-tal kapott eredmények azonban sokkal tömörebbek és könnyebben érthetők voltak. A hibakereső rendszer használata megköveteli egy tanár jelenlétét.[2] A tanuló algoritmus hatékonyan fel tudta használni a tanár által a tanulás közben pótlólagosan bevitt információkat a hipotézis előállításához.

Az IMPUT rendszer IDTS hibakereső modulját a szerző és munkatársai közösen készítették az „ILP" BRA 6020 ESPRIT projektben. Segítségével az aktuális hipotézist és a tanulási példákat felhasználva lokalizálható a hiba helye, ami miatt az aktuális hipotézis nem viselkedik megfelelően. Az aktuális hipotézist az adott helyen megváltoztatva egy újabb hipotézis kapható. A transzformáció sorozat bizonyos feltételek esetén bizonyíthatóan véget ér, sőt el is jut egy, a példákkal konzisztens hipotézishez. Léteznek azonban olyan tanulási példák, amelyek esetében az algoritmus nem tud eljutni a megfelelő hipotézishez. Egy elméleti eredmény szerint [Chen96] létezik teljes megoldás, olyan, amely tetszőleges tanulási példa halmaz esetén el tud jutni egy megfelelő hipotézishez, azonban a megoldás módjára ez az elméleti eredmény nem ad választ.

A 4. Fejezetben több a természetesnyelv-feldolgozáshoz kötődő alkalmazásról esik szó. A szerző vezette be a természetes nyelvi egyértelműsítési feladat megoldásában a bizonytalansági-osztály fogalmát. Saját eredmény a bizonytalansági osztályokra alapozott tanulási modell kidolgozása több tanuló algoritmusra (C 4.5, Progol, AGLEARN). A szerző dolgozta ki az IMPUT algoritmus alkalmazási modelljét a magyar magánhangzók felismerésére. A THALES természetes nyelvi interfész attribútum nyelvtan alapú objektum memóriájának kifejlesztése is a szerző saját munkája.

A beszédfelismerésben a tanuló algoritmusok feladata könnyen definiálható: adott néhány hang minta, amelynek a szöveg megfeleltetése ismert. A feladat egy ismeretlen digitalizált hullám szöveges megfelelőjének előállítása. Ebben a körben egy igen fontos feladat a folyamatosan érkező hullámok szegmentálása, azaz elemi építő elemekre (fonémákra) bontása. A következőkben az egyes fonémák azonosítása a cél, amely feladatra a különböző tanuló algoritmusok alkalmazása magától értetődik. Statisztikus módszereket már régóta használnak beszédfelismerésre [Jeli97], [Rab93]. A magyar magánhangzók akusztikus szempontból viszonylag egyszerűek, néhány domináns spektrális összetevő szuperpozíciójának tekinthetők. Természetesen a frekvencia és intenzitás értékek a beszélőtől függnek, továbbá ezek az értékek időben is változhatnak. A szerző egy tanulási modellt dolgozott ki, amely az IMPUT tanuló algoritmus alkalmazására épült. A rendszer szabály-halmazokat tanult meg öt magyar magánhangzó azonosítására. További magánhangzók felismerésére a rendszer alkalmassá tehető, mássalhangzók esetében azonban további akusztikus tulajdonságok beépítése szükséges.

A természetes nyelvekkel kapcsolatos kutatások egyik ága a természetes nyelvi interfészek. Az ilyen típusú programok informatikai előképzettség nélküli természetes nyelven történő kommunikációt biztosítanak a felhasználó és a számítógépek között. A természetes nyelvi interfészek nem engedik meg a teljes természetes nyelv használatát, annak csak egy jól definiált részhalmaza használható. A mondatok egy jól meghatározott területhez kapcsolódó objektumainak leírására és a rajtuk értelmezhető tevékenységek elvégzésére utasító parancsok kiadására alkalmasak. A gyakorlatban is jól működő rendszer egyik kulcsa a pontosan definiált parancs szintaxis és az erre épülő szemantika. A szerző és munkatársai egy attribútum nyelvtan alapú, nagymértékben általánosítható interfész generátor rendszert készítettek, amely síkgeometriai szerkesztések végrehajtására alkalmas. A szerző saját eredménye az objektumokat tároló memória, a szimbólumtábla attribútum nyelvtanos specifikáción alapuló előállítása volt, továbbá a kezeléshez szükséges eljárások forráskódjának generálása.

A természetes nyelvek feldolgozása közben több egymástól jól elkülöníthető feladatot definiáltak [Man00]. A feldolgozás a szöveg strukturális felbontásával kezdődik, az adatállományokban található szövegeket fejezetekre, bekezdésekre, mondatokra és szavakra bontják fel. Már ebben a lépésben is több gyakorlati szempontból lényeges probléma merül fel (mondathatárok és a szavak határainak meghatározása, idézőjelek, kötőjelek hatókörének kijelölése). A továbbiakban fontos feladat a szavak szófaji elemzése. A szófaji kódokat egy címkével csatolják a szavakhoz. A természetes nyelvekben gyakran fordulnak elő többjelentésű szavak. A többjelentésű szavak címkéi közül a szövegkörnyezetnek megfelelő címke kiválasztását nevezik **szófaji egyértelműsítésnek**. Az egyértelműsítési feladat megoldására már régóta alkalmaznak statisztikai (HMM) tanuló algoritmusokat [Cut92], [Mer94]. Az utóbbi időben több szabályalapú megközelítést is publikáltak

[2] A szakirodalomban a tanárt sokszor nevezik *bölcsnek* (oracle). A továbbiakban a dolgozatban is ez az elnevezés fordul elő.

[Cuss97], [Eine99]. A tanuló algoritmusok alkalmazásakor nehézséget jelent a tanulási példák nagy száma. A szerző kidolgozott egy a gyakorlatban is hatékonyan működő módszert a tanulási példák számának csökkentésére azáltal, hogy a tanulási feladatot sikeresen fel tudta bontani kisebb feladatokra. Az eredményül kapott szabályhalmazok egyesítésével lehet a végeredményt megkapni. A szerző által kidolgozott módszer alapján több különböző tanuló algoritmus felhasználásával határoztak meg egyértelműsítési szabályokat, amelyeket egyértelműsítő programokba építettek be. A szerző által kidolgozott elvek alapján készültek el a C 4.5, a Progol és az AGLEARN algoritmusok alkalmazásai. A szerző munkatársaival egy összehasonlító tanulmányt készített öt különböző tanuló algoritmus felhasználásával, továbbá sikeres kísérleteket végeztek a különböző szabályrendszerek kombinálására.

A természetes nyelvek feldolgozásának következő lépéseiben [Gin93] – főnévi szerkezetek kijelölése (NP-chunking) [Joh00], [Zhou00], szintaktikus elemzés [Dej00], szemantikus annotáció, anafora feloldás (anaphora resolution)[3], szándék felismerés, szöveg kategorizálás [Seb02], szöveg kivonatolás [Man99] – ugyancsak alkalmas területei a tanuló algoritmusok alkalmazásának.

A dolgozat elkészítéséhez felhasznál publikációk megjelenését követően a szerző az IKTA 27/2000 projekt egyik irányítójaként részt vett egy magyar nyelvű 1 millió szavas tanuló adatbázis fejlesztésében. Az adatbázis jelentős emberi erőforrások bevonásával 2002-ben készült el. A szerző és munkatársai kísérleteket kezdtek tanuló algoritmusok futtatására és egyértelműsítési szabályok tanulására. A megtanult szabályok segítségével gyakorlatban is működő egyértelműsítő rendszert készítettek. Az NKFP 02/17/2001 projektben a névszói szerkezetek struktúráját vizsgálják, amelynek során a főnévi szerkezet nyelvtanának tanulására az IMPUT rendszert fogják alkalmazni.

Az Európai Közösség két ízben is támogatott egy olyan kutatási projektet, amely témája az ILP (Inductive Logic Programming) a logikai programok (elsőrendű logikai formulák) tanulása volt. Az első a BRA (Basic Research Area) 6020 „ILP" ESPRIT projekt az elméleti alapok lerakására irányult 1993–1996 között. A második pedig az LTR (Long Term Research) 20237 „ILP2" ESPRIT projekt az ILP tanuló algoritmusok lehetséges gyakorlati alkalmazási területeinek kijelölésére irányult 1996–1999 között. Mindkét projektben 8 európai egyetem vett részt, valamint több társintézmény és ipari partner. A Szegedi Tudományegyetem társintézmény volt mindkét projektben. A szerző tevékenyen részt vett a projektek kutatási-fejlesztési munkáiban.

A magyar természetes nyelvi kutatásokat nagy mértékben segítette, a magyar Oktatási Minisztérium IKTA és NKFP kutatás-fejlesztési pályázatainak meghirdetése. A szerző és társai által beadott nyertes pályázatok: IKTA 27/2000, NKFP 2/017, IKTA 37/2002. A pályázatok során több a magyar természetes nyelvhez kapcsolódó kutatás indult el. A szerző több más kisebb projektben is részt vett többek között: ILPNet Copernicus CP-43, ILPNet2 EU INCO program 9771002, PHARE TDQM H9305-02/1022, valamint több OTKA pályázat.

Szeretném kifejezni köszönetemet Gyimóthy Tibornak, témavezetőmnek, aki pályakezdésem óta munkatársam és támogatóm volt, aki nélkül a dolgozat nem készülhetett volna el, Horváth Tamásnak és Kókai Gabriellának, akikkel sok éven át együtt dolgoztam együtt közös kutatási feladatokon és Csirik Jánosnak az Informatikai Tanszékcsoport vezetőjének, aki szakmai tanácsain túl nagymértékben hozzájárult a kutatási feltételek megteremtéséhez.

Ugyancsak köszönettel tartozom az Informatikai tanszékcsoport dolgozóinak, munkatársaimnak, akik ösztönöztek, segítettek és támogattak a dolgozat megírásában és családom tagjainak, akik már nagyon várták a dolgozat elkészültét.

[3] Anafórának nevezik az azonos objektumokra történő különböző referenciákat. A természetes nyelvekben személyekre, tárgyakra gyakran hivatkoznak rokon értelmű kifejezésekkel pl. „János beindította az autót. Az Fiat lassan elindult." – az előbbi két mondatban az autó és a Fiat ugyanazt az objektumot jelöli. Bonyolultabb esetén a feloldás háttértudást is igényelhet pl. Bush találkozott a katonákkal. A hadsereg főparancsnoka együtt ebédelt a tisztekkel." – itt tudni kell, hogy Bush az Egyesült Államok elnöke, aki egy személyben a hadsereg főparancsnoka is.

2. Szabályalapú gépi tanulási módszerek

A tanuló algoritmusok alkalmazásai általában két különböző diszciplína közötti együttműködés keretében készülnek. Az egyik fél a mesterséges intelligenciával foglalkozó, tanuló algoritmusokhoz értő szakemberek csoportja, a másik pedig az adott megoldandó (orvosi, biokémiai, mérnöki, fizikai stb.) feladat szakértőiből álló csoport. A közös munka szempontjából kritikus, hogy milyen interfészt használva kerülhetnek be a számítógépes rendszerbe egy szakma speciális ismeretei, háttér tudása, illetve tapasztalati tényei (pl. mérési eredményei). A szabályalapú rendszerek biztosítani tudják azt az elegendően flexibilis hátteret, ami az adatok és a kiszámítási szabályok egységes alakra hozásához szükséges.

A szabályalapú rendszerek egy fontos osztályt alkotnak a tanuló algoritmusok között. Elterjedésüket előnyösen befolyásolja, hogy a bennük alkalmazott szabályok az ember számára közvetlenül olvashatók és megérthetők. Hátrányként szokás megemlíteni, hogy az ilyen típusú rendszerek létrehozása nem teljesen automatikus, illetve, hogy a találati pontosság is sok esetben rosszabb mint például a neurális-hálózatokon alapuló módszerekkel elért eredmények.

Amikor rögzítésre kerül egy tanuló algoritmus hipotéziseinek tere, az egyúttal meghatározza a rendszerbe bekerülő tanulási példák, a háttér tudás és a hipotézisek lehetséges szerkezetét, szintaxisát. A formálisan rögzített szintaxis lehetővé teszi, hogy olyan algoritmusokat készítsünk, amelyek szisztematikusan bejárják a hipotézisek terét és a példákat figyelembe véve egy alkalmasabb új hipotézisekre találjanak. A generatív nyelvtanok témakörébe tartozó ismeretek alkalmas hátteret biztosíthatnak ehhez a munkához.

A bevezetőben adott 1.1. definícióban nem volt elkülönítve a teszt adatokon számításokat végző Z program és az az algoritmus, amely a Z-t esetleg módosítja. Ez szabályalapú rendszerek esetében megtehető, sőt kívánatos is:

2.1. Definíció: Adott egy H hipotézis tér, E tanulási példák egy halmaza, T a teszt feladatok egy halmaza, egy P végrehajtási teljesítmény mérték, valamint egy $Z_0 \in H$ kiindulási hipotézis és B háttér tudás (amely lehet az üres halmaz is). Az L tanuló algoritmus feladata egy olyan Z_1 hipotézis keresése a H hipotézis térben az E tréning példákat figyelembe véve, hogy a $(Z_1 \cup B)$ hipotézis P mérték szerinti teljesítménye jobb legyen, mint a $(Z_0 \cup B)$ hipotézisé a T teszt példákon.

A fenti definíció szerint az L tanuló algoritmus bemenő adatként kezeli a Z_0 hipotézist, majd azt módosítva állítja elő az újabb Z_1 hipotézist. A fenti definíció egyelőre nem tér ki arra, hogy vajon létezik-e ilyen Z_1 hipotézis, mi történik akkor, ha belső ellentmondás van a tréning vagy a teszt adatok között, vagy a háttér tudás helytelen. Nyitott kérdés továbbá, hogy hogyan lehet az új Z_1 hipotézist előállítani: a korábbi Z_0 egyik szabályának módosításával, vagy egy teljesen új szabály hozzávételével?

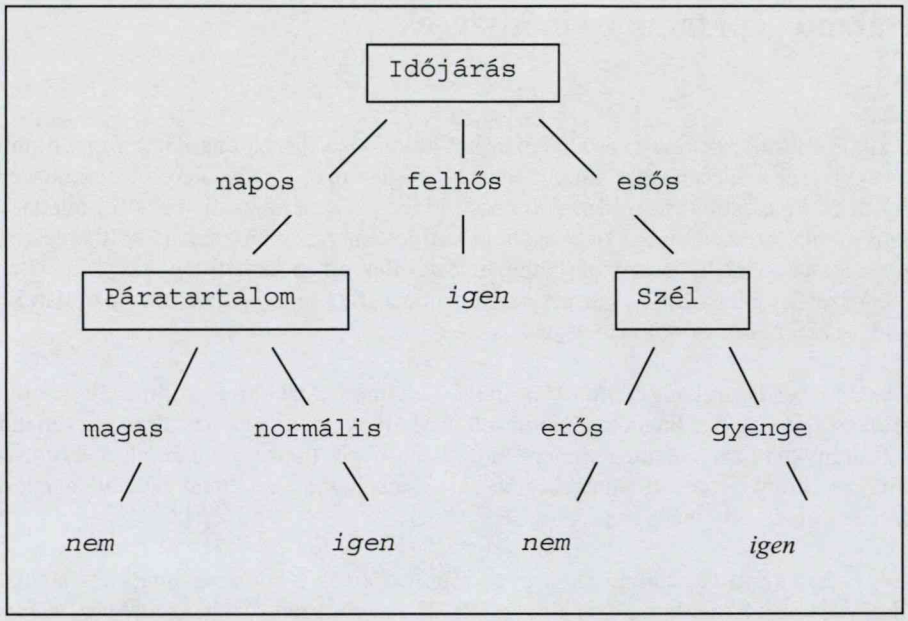
A következőkben ismertetésre kerül a tanuló algoritmusok egyik korai változata, az ID3. A szakirodalom némileg megosztott abban a tekintetben, hogy ez a módszer valóban szabályalapú-e. Egy a későbbiekben ismertetésre kerülő algoritmussal a döntési fák egyszerűen átalakíthatók *„ha-akkor"* típusú szabályokká, ezért szerepel joggal az ID3 algoritmus a szabály-alapú tanuló algoritmusok között. Az is indokolja a szerepeltetését, hogy az ID3 alapul szolgált több később ismertetésre kerülő tanuló algoritmushoz.

2.1. Döntési fák tanulása, az ID3 algoritmus

Az ID3 algoritmust az ausztrál Quinlan publikálta 1986-ban [Quin86]. Később ennek felhasználásával készítette el munkatársaival a C 4.5 tanuló rendszert [Quin93]. Az ID3 algoritmus Mitchel [Mit97] könyvének 3. Fejezetében leírtak alapján kerül bemutatásra.

Az ID3 algoritmus döntési fák tanulására szolgál. A döntési fák tekinthetők a diszkrét változókon értelmezett diszkrét értékű függvények egy reprezentációjának. Később az ID3 algoritmust kiterjesztették folytonos attribútum értékekre [Fay92]. A döntési fákban három lényeges objektum különböztethető meg: levelek (a levél címkék adják meg a függvény értékét), belső pontok (a belső pontok címkéi a függvény egy-egy bemenő változóját *attribútumát* jelölik ki) és csúcsok között élek (az élek címkéi a szülő csúcsban adott

attribútum egy-egy lehetséges értékét tartalmazzák).



2.1. ábra: Példa döntési fára

A 2.1. ábrán bemutatott döntési fa egy olyan diszkrét értékű függvényt reprezentál, amelynek a lehetséges értékei logikai típusúak (*igen*, *nem*). A függvénynek legalább három bemenő változója van: az időjárás (lehetséges értékei: *napos*, *felhős*, *esős*), a páratartalom (lehetséges értékei: *magas*, *normális*) és a szél (lehetséges értékei: *erős*, *gyenge*). Egy adott napon amikor ismert a (várható) időjárás, a páratartalom és a szél paraméterek értéke a döntési fa segítségével kiszámítható függvény értéke (az időjárás alkalmas-e teniszezésre).

Nap	Időjárás	Hőmérséklet	Páratartalom	Szél	Függvényérték (alkalmas teniszezésre)
D1	Napos	Meleg	Magas	Gyenge	Nem
D2	Napos	Meleg	Magas	Erős	Nem
D3	Felhős	Meleg	Magas	Gyenge	Igen
D4	Esős	Enyhe	Magas	Gyenge	Igen
D5	Esős	Hűvös	Normális	Gyenge	Igen
D6	Esős	Hűvös	Normális	Erős	Nem
D7	Felhős	Hűvös	Normális	Erős	Igen
D8	Napos	Enyhe	Magas	Gyenge	Nem
D9	Napos	Hűvös	Normális	Gyenge	Igen
D10	Esős	Enyhe	Normális	Gyenge	Igen
D11	Napos	Enyhe	Normális	Erős	Igen
D12	Felhős	Enyhe	Magas	Erős	Igen
D13	Felhős	Meleg	Normális	Gyenge	Igen
D14	Esős	Enyhe	Magas	Erős	Nem

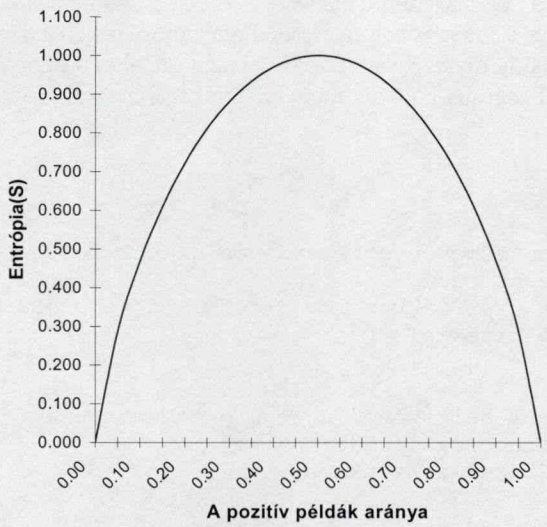
2.2. táblázat tanulási példák az ID3 algoritmushoz

A függvény értékét a fa egy levelének címkéje tartalmazza. Ehhez a levélhez a következőképpen lehet eljutni: a fa gyökerétől elindulva mindig olyan éleken kell haladni, amelyek címkéi azonosak a függvény argumentumaiban megadott attribútumok értékeivel. Ha például az adott napon az időjárás *esős*, a szél pedig *gyenge*, akkor a gyökérből elindulva a harmadik élet (*esős*) kell választani, majd a „Szél" csúcsból tovább a *gyenge* címkéjű élen az *igen* értékhez érkezünk el.

A döntési fák tanulásakor egy táblázatban megadjuk az ismeretlen függvény értékét néhány pontban, majd a tanuló algoritmus feladata egy olyan döntési fa elkészítése, amely a táblázatban megadott helyeken az ott

megadott értékeket számítja ki. Az ID3 algoritmusnak nincs szüksége kezdeti döntési fára, minden esetben egy teljesen új fát készít a bemenő adatok függvényében.

A 2.2. táblázatban található egy adatsor, amelyen bemutatásra kerül az ID3 algoritmus működése. Az nyilvánvaló, hogy ismerve az összes döntési fát tartalmazó keresési teret, egy nem túl bonyolult programmal különféle döntési fák állíthatók elő, amelyek konzisztensek a megadott adatokkal, illetve szisztematikus bejárással ki lehet választani ezek közül a legegyszerűbb döntési fát. A Quinlan által felfedezett módszer legnagyobb előnye az, hogy az input adatok feldolgozásával egy lépésben elő tudja állítani a döntési fát.



2.3. Ábra. Az *entrópia* a pozitív példák arányának függvényében

Az algoritmus szempontjából a legfontosabb lépés a döntésbe bevonni kívánt attribútum kiválasztása. A kiválasztási stratégia az **entrópia** kiszámítására épül. A teljes példahalmaz (S) entrópiáját a lehetséges függvényértékek vizsgálata alapján tudjuk kiszámítani. Az entrópiát szokás *bonyolultsági mértéknek* (impurity measure) is nevezni. Ez a mennyiség a SPECTRE és az IMPUT algoritmusokban is szerepet játszik majd.

$$Entrópia\ (S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

A képletben a c a lehetséges függvényértékek száma (pl. igen/nem esetén 2). A p_i pedig az i-ik függvény értékhez tartozó példák száma az összes példa számával elosztva. A tekintett példában 9 esetben a függvény értéke *igen* volt és 5 esetben *nem*. Az entrópia tehát:

$$Entrópia\ (S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0{,}940$$

Intuitíve, az entrópia azt a szükséges információ mennyiségét (a bitek számát) jelöli, ami ahhoz kell, hogy a véletlenszerűen érkező példákhoz hozzárendeljük a függvényértékeket. Ha a jelen esetben nem lenne olyan adat, amelyhez a függvény értéke *igen*, akkor az entrópia *0* lenne, ami azt jelenti, hogy mivel ilyenkor minden példához ugyanaz a konstans érték tartozik (*nem*), a szükséges bitek száma nulla. Amennyiben az *igen* értékhez és a *nem* értékhez is ugyanannyi példa tartozott volna, úgy az entrópia *1.0* lett volna, ami az mutatja, hogy ekkor pontosan egy bitre van szükség ahhoz, hogy minden példára megadjuk a függvény értékét (*igen/nem*).

A továbbiakban fontos még az információ nyereség fogalma. Ez azt mutatja meg, hogy egy A attribútum mennyire hatékony a példák szétválogatásakor. A példák megcímkézéséhez esetleg kevesebb bit is elegendő, ha ismerjük minden példában egy A attribútum értékét. Az eddigi példánál maradva, ha mondjuk az A attribútum v_1 és v_2 értéke két részre vágná a példákat és mindegyik részbe már csak olyan példák kerülnének, amelyek esetén a függvény értéke azonos lenne, akkor ezt az ismeretet felhasználva a szükséges bitek száma nullára csökken (az A attribútum értékeit maguk a példák tartalmazzák, az onnan kiolvasható).

$$Nyereség(S,A) = Entrópia(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entrópia(S_v)$$

A *Values(A)* az *A* attribútum lehetséges értékeinek halmazát, az abszolút érték pedig az adott halmaz elemeinek számát jelöli. Az S_v halmaz azokat a példákat jelöli ($\subseteq S$), amelyekben minden elemre az *A* attribútum értéke éppen *v*. Válasszuk ki most a „Szél" attribútumot, az S_{gyenge} halmazba 8 példa tartozik (6 esetben *igen* és 2 esetben *nem* a függvény értéke, az Entrópia(S_{gyenge}) = 0.811), az $S_{erős}$ halmazba 6 példa tartozik (3 esetben *igen* és 3 esetben *nem* a függvény értéke, az Entrópia($S_{erős}$) = 1.0). Az információ nyereség:

$$Nyereség(S, Szél) = Entrópia(S) - (8/14)0{,}811 - (6/14)1{,}0 = 0{,}048$$

Az „időjárás" nyeresége 0,246, a „páratartalom" nyeresége 0,151, a „hőmérséklet" nyeresége pedig 0,029. Természetesen a döntési fa csúcsához a legnagyobb nyereségű attribútumot kell hozzárendelni. Az algoritmus ezután rekurzívan addig folytatódik, amíg olyan S_v halmazokig el nem jut, ahol a függvényértékek már azonosak. Ekkor a fa készítése véget ér, a levél csúcsba a közös függvényértéket írja az algoritmus. A teljes algoritmus a következő:

```
ID3(Példák, Attribútumok)
begin
  Egy Gyökér csúcs létrehozása
  if a példa halmazban minden függvényérték azonos, akkor levél csúcs létrehozása,
    a közös értékkel és visszatérés
  if az Attribútumok halmaza üres, akkor levél csúcs létrehozása a példák között
    leggyakrabban előforduló értékkel
  otherwise
  begin
    Legyen A = az Attribútumok halmazából a legnagyobb nyereségű
    Belső (döntési) pont létrehozása a fában A-val
    forall v lehetséges értékére az A-nak do
    begin
      új fa csúcs létrehozása, az élcímke v legyen
      P[v] legyen a Példák halmazából kiválogatott elemek halmaza, amelyekben az
           A attribútum értéke v.
      if a P[v] üres, akkor
        Egy levél csúcs létrehozása a Példák halmazban előforduló leggyakoribb
        függvényértékkel
      egyébként
        ID3(P[v], Attribútumok - {A})
    end forall
  end otherwise
end ID3;
```

Az algoritmust lefuttatva a 2.2. táblázatban felsorolt adatokra a 2.1. ábrán bemutatott döntési fát kapjuk. Az ID3-nak számos előnyös tulajdonsága van. Egyik például az, hogy a példákban nem szükséges ismernünk minden attribútum értékét. Látható az eredményen, hogy például a hőmérséklet nem szerepel a végeredményül kapott döntési fában, ezért ha egyes példákban a hőmérséklet nem lett volna ismert, az nem okozott volna problémát, ugyanezt a fát kaptuk volna. Az algoritmus azt is elviseli, ha a példák között egymásnak ellentmondó adatok fordulnak elő. Ebben az esetben a függvény értéke az lesz, amely a példákban a legtöbbször fordul elő (a fenti algoritmusban a második „`if`" utasítás).

2.1.1. A döntési fával ekvivalens szabályrendszer

A döntési fák ekvivalens „ha-akkor jellegű" szabályokká is könnyen átalakíthatók a következőképpen: ahány különböző levél van a döntési fában, annyi feltételes utasítást készítünk, amelynek a feltétel részéhez a levélbe vezető útnak megfelelő logikai formulát rendeljük (a szintenkénti relációkat az *és* logikai művelettel kapcsoljuk össze). A 2.1. ábrán bemutatott döntési fában öt levél van, ezért a vele ekvivalens feltétel rendszer a következő lesz:

```
if (időjárás = napos) and (páratartalom = magas) then func = nem ;
if (időjárás = napos) and (páratartalom = normális) then func = igen ;
if (időjárás = felhős) then func = igen ;
if (időjárás = esős) and (szél = erős) then func = nem ;
if (időjárás = esős) and (szél = gyenge) then func = igen ;
```

A fenti feltételrendszer különböző módokon egyszerűsíthető, például az azonos függvényértékekhez tartozó feltételek egyesíthetők (a logikai *vagy* művelettel):

```
if ((időjárás = napos) and (páratartalom = magas)) or
   ((időjárás = esős) and (szél = erős)) then func = nem ;
```

illetve

```
if ((időjárás = napos) and (páratartalom = normális)) or
    (időjárás = felhős) or
   ((időjárás = esős) and (szél = gyenge)) then func = igen ;
```

2.1.2. Az ID3 algoritmus helye a tanuló algoritmusok között

Az ID3 algoritmus a tréning példák attribútumait egyszerűeknek, egymástól függetlennek tételezi fel. Strukturált adatokat az ID3 algoritmus nem tud kezelni. Ezt a fajta tanulást attribútum-érték tanulásnak nevezik (attribute-value learning). Az elnevezés a tréning példák egyszerűségére utal. A tanulás során kitüntetett szerepe van annak az esetnek, amikor a megtanulandó függvény logikai értékű (igaz/hamis).

2.2. Definíció [Mit97]: Fogalom tanulásnak nevezzük az olyan tanulási feladatot, amelyben a feladat egy logikai (boolean) értékű függvény megtanulása az input és output értékeket tartalmazó példák alapján.

A 2.2. Definícióban az absztrakciót még tovább növelve azt kapjuk, hogy egy tetszőleges U halmaz (egy univerzum) egy C ($\subseteq U$) részhalmaza reprezentál egy fogalmat. A tanulás során meg kell határozni logikai formulák egy rendszerét, amely tetszőleges $u \in U$ elemre megadja, hogy az u elem benne van-e C-ben. Egy halmaz mértéket bevezetve a megtanult logikai rendszer hibája is jól definiálható, például annak a halmaz résznek a mértékével, ahol a formulák helytelen eredményt adnak. (Hiba két esetben is előfordulhat: helytelenül állítja a rendszer, hogy egy u elem benne van C-ben, illetve ha a rendszer helytelenül állítja azt, hogy nincs benne.)

A formális logikában már régóta ismert az egyszerű logikai változókat és logikai műveleteket (és, vagy, tagadás, implikáció, ekvivalencia) tartalmazó formulák osztálya. Ezeket a formulákat *propozícióknak* nevezik, a kutatási területet pedig *propozícionális logikának* [Ner97]. A számítógépes tanulásnak azt a részterületét, amikor a hipotézistér a propozíciók tere, *propozícionális tanulásnak* (propositional learning) nevezik [Lav94]. A 2.1.1. Fejezetben bemutatott konverziós módszerrel az ID3 döntési fái propozíciókká alakíthatók. Ezért az ID3 tulajdonképpen egy *propozíciós formula tanuló algoritmus*, vagy rövidebben *propozícionális tanuló algoritmus*.

Az ID3 algoritmus kapcsán még egy jelenségre érdemes kitérni – ez a túlspecializálás. Akkor fordul elő, ha a tréning példákból kiindulva a megtanult hipotézist a „végletekig" finomítjuk.

2.3. Definíció: A túlspecializálás [Mit97]. Adott egy H hipotézis tér. Egy $h \in H$ hipotézis túlspecializált a tréning adatokra, ha létezik egy olyan $h' \in H$ hipotézis, hogy a h hipotézisnek kisebb a hibája a tréning adatokon mint h'-nek, viszont a h' hipotézisnek kisebb a „várható" hibája a teljes alaphalmazon, mint h-nak.

A tanulási feladatok túlnyomó részében van valamekkora hiba. A tanulási adatbázisban a legkülönbözőbb okok miatt fordulhatnak elő hibák. Bonyolult fogalom esetén a tanuló rendszer is csak olyan hipotéziseket talál (a rendelkezésére álló idő alatt), amelyek csupán közelítőleg felelnek meg a példáknak. Egy partikuláris példának túlzott jelentőséget tulajdonítani azzal a veszéllyel jár, hogy a hipotézis viselkedése éles adatokon romlik. Az ID3 esetében a túlspecializálás elkerülésére több módszert fejlesztettek ki, amelyek a megtanult döntési fa egyszerűsítésén (nyesésén, pruning) alapulnak. Az algoritmus először mindenképpen elkészíti (az esetleg túlspecializált) teljes döntési fát. A következő lépésben a döntési fát feltételes utasítások rendszerévé alakítja (lásd 2.1.1. Fejezet). Ez után a szabályok fejében előforduló relációk közül egyet-egyet törölnek[4], amennyiben az így kapott új szabály pontossága növekszik egy ún. validációs halmazon. A validációs halmaz lehet a tréning példák egy elkülönített része, vagy egy külön erre a célra megadott adatsor. Amikor már újabb relációkat elhagyni nem lehet a feltételes utasítások fejéből, akkor az algoritmus véget ér. A végeredményhez még utoljára sorba rendezik a szabályokat – a pontosság szerint legjobb szabály kerül előre.

[4] Ezt a tevékenységet felfoghatjuk a szabály általánosításának.

2.2. *Logikai programok tanulása (ILP)*

A formális logikában felállított logikai nyelv hierarchia szerint a propozíciók azonosak a 0-ad rendű logikai nyelvvel leírható formulákkal. A velük foglalkozó területe a logikának a *0-ad rendű logika*, más néven az *ítéletkalkulus*. Mivel a propozícionális logikában nincsenek kvantorok, predikátumok és függvényszimbólumok, ezért a gyakorlati felhasználhatósága korlátozott. Az elsőrendű logikában megtörténik ezek bevezetése, aminek köszönhetően az elsőrendű logikai formulák leíró ereje ugrásszerűen megnő. A gyakorlati alkalmazások szempontjából az elsőrendű formuláknak kimagasló jelentőségük van, mivel a valóságban előforduló törvényszerűségek, jelenségek sok esetben jól modellezhetők elsőrendű elméletekkel. Az elsőrendű logikára szokás *predikátumkalkulusként* is hivatkozni. Az ilyen formulák tanulását pedig *predikátum tanulásnak* (predicate learning), *elsőrendű tanulásnak* (first order learning), vagy *relációs tanulásnak* (relational learning) nevezni.

2.4. Definíció: Elsőrendű logika körébe tartozó alapfogalmak [Lav94]:
a.) *elsőrendű abc.* Az elsőrendű abc három véges halmazból áll: a változók, a függvény szimbólumok és a predikátumok halmazából. Az utóbbi kettő elemei *rangoltak*, azaz minden függvény és predikátum szimbólumhoz tartozik egy *aritás* érték, amely a lehetséges argumentumaik számát jelöli. A *0* aritású függvény szimbólumokat nevezzük konstansoknak.
b.) *term.* Minden változó és konstans term. Ezeken kívül minden esetben ha *f/n* egy *n* aritású függvény szimbólum és $t_1, t_2, ..., t_n$ termek, akkor az $f(t_1, t_2, ..., t_n)$ is term.
c.) *atom.* Minden esetben, ha p/*m* egy *m* aritású predikátum szimbólum és $t_1, t_2, ..., t_m$ termek, akkor a $p(t_1, t_2, ..., t_m)$ atom. A változó nélküli atom neve *tényállítás* (fact).
d.) *literál.* Egy atomot vagy annak a negáltját literálnak nevezik. A kettő között különbséget lehet tenni: *pozitív literál* az atom, negatív literál a negált atom neve.
e.) *klóz.* Literálok egy halmaza például $\{L_1, L_2, \neg L_3, \neg L_4, \neg L_5\}$, Ez alatt a következő logikai formulát kell érteni: $\forall x_1 \forall x_2 ... \forall x_i (L_1 \vee L_2 \vee \neg L_3 \vee \neg L_4 \vee \neg L_5)$. Az $L_1, ..., L_5$ literálokban előforduló összes változó a $\forall$ kvantorral (bármely) le van kötve. Ez a formula ekvivalens a $\forall x_1 \forall x_2 ... \forall x_i (L_1 \vee L_2 \leftarrow L_3 \wedge L_4 \wedge L_5)$ formulával. Ennek a jelentése az, hogy minden esetben, amikor a jobb oldalon álló $L_3 \wedge L_4 \wedge L_5$ kifejezés igaz, abból következik, hogy az $L_1 \vee L_2$ kifejezés is igaz. Az $L_3 \wedge L_4 \wedge L_5$ kifejezést a klóz *törzsének*, az $L_1 \vee L_2$ kifejezést a klóz *fejének* hívják.
f.) *Horn-klóz*: A legfeljebb egy pozitív literált tartalmazó klóz. Ez tartalmazza azt az esetet, amikor a klóz 1 pozitív literált tartalmaz (ez a definit klóz), egyetlen pozitív literált sem tartalmaz (ennek a neve definit cél/goal). Ha a klóz csak egyetlen pozitív literált tartalmaz, akkor azt egység (unit) klóznak nevezik.
g.) *(Prolog) program klóz.* Amennyiben a formula $\forall x_1 \forall x_2 ... \forall x_j (H \leftarrow L_1 \wedge ... \wedge L_m)$ alakú, ahol minden L_i literál vagy egy atom, vagy egy atom negáltja. Az *m* értéke lehet *0* is (unit klóz).
h.) *Normál (Prolog) program.* Program klózok egy halmaza.
i.) *Predikátum definíció.* Program klózok egy halmaza, amelyeknek a fejében ugyanaz a predikátum található.

A logikai programozási nyelv – a Prolog – szintaxisa az elsőrendű logikai nyelv egy bővítésének felel meg, amennyiben a klózok törzsében negatív literál is előfordulhat (2.4. Definíció, g.) pontja). A Prolog nyelven készült programokhoz hozzárendelhető egy kiszámítási szemantika, amely alkalmazásával a Prolog nyelv bármely imperatív programozási nyelv helyébe léphet. Egy tetszőleges imperatív programozási nyelven készült programnak mindig elkészíthető a Prolog nyelvű változata.

Több esemény is hozzájárult ahhoz, hogy a szabályalapú tanuló programok között az elsőrendű logikai formulák tanulása végül kiemelkedően fontossá vált. Az egyik ok minden bizonnyal a Prolog programozási nyelvnek, mint a mesterséges intelligencia programozási nyelvének az előretörése volt, valamint az, hogy az elsőrendű logika nyelve sokkal jobban meghatározott (a Prolog nyelv is szabványosított), mint mondjuk az imperatív programozási nyelvek sokasága, nem is beszélve a számtalan verzióról. Ezen kívül a Prolog nyelv számos olyan szolgáltatást nyújt, amelyet az általánosan használt programfejlesztési környezetek nem tudnak: fák, listák kezelése, önmagukat módosító programok, metainterpreter[5]. Az első kezdeti lépések akkor történtek meg, amikor Shapiro [Shap83] a Prolog programok hibakeresésével foglalkozva rájött, hogy módszere a programok automatikus megváltoztatására is lehetőséget nyújthat.[6] Az általa elkészített MIS rendszer számos

[5] A felhasználói programokból is meghívható a Prolog interpreter, az eredmény lekérdezhető.

[6] A Prolog programozási nyelv lehetőséget nyújt a programok számára ahhoz, hogy futás közben önmagukat módosítsák. Ezt az teszi lehetővé, hogy a formuláknak van egy viszonylag könnyen kezelhető fa reprezentációja. A fát egy Prolog változóba

azóta készített elsőrendű tanuló rendszer alapja lett.

Shapiro a Prolog programok lehetséges hibáit négy csoportba sorolta, az egyes hibatípusok lokalizálására egy hibakereső rendszert készített. A négy típus a következő: hibás eljárás (false procedure), nem értelmezett input (uncovered goal), végtelen ciklus (looping), divergencia (diverging). A rendszer alapja egy mélységben korlátozott metainterpreter, amely kérdéseket tesz fel a hibát kereső ember számára. Ezt a személyt tanárnak, illetve bölcsnek (oracle) nevezte el. Amikor a tesztelő rendszer a számos teszt esetet ellenőrzi, és véletlenül egy hibára bukkan, akkor a hibakereső rendszer aktiválódik. Ez után a *bölcshöz* intézett kérdésekkel a program megkeresi a hiba forrását, egy logikai formulát. A MIS rendszer képes volt a megtalált formulát megváltoztatni, az így kapott hipotézist ismét ellenőrizni a teszt példákon, hiba esetén újból megkeresni a hiba helyét, újból javítani és így tovább. Ezt a tevékenységet addig ismételte a rendszer, amíg csak az összes teszt példa hibátlanul le nem futott a folyamatosan változó programmal. A MIS rendszer bizonyította a működőképességét kis példákra. Számos kombinatorikus robbanási hely volt azonban a rendszerben: gyorsan szélesedő keresési tér, a hibakereséshez szükséges kérdések száma, amit a *bölcsnek* meg kellett válaszolnia.

A logikai programozás egyik kulcs objektuma a predikátum. A predikátumok logikai értékű függvények, amelyek tetszőleges halmazokon vannak értelmezve. Az értékük *igaz* vagy *hamis*. Láthatóan éppen megfeleltethetők a fogalom tanulásban keresett ismeretlen szabályrendszernek, amely az ismeretlen fogalmat hivatott leírni. A tanulási példáknak tartalmazniuk kell azt is, hogy rajtuk az ismeretlen predikátum értéke *igaz* vagy *hamis* kell legyen. Mivel itt csak ez a két eset fordulhat elő, ezért a leggyakrabban a példák halmazát két részre bontják: *pozitív példák* halmazára (itt a megtanulandó predikátum értéke igaz kell legyen) és *negatív példák* halmazára (itt a megtanulandó predikátum értéke hamis kell legyen).

A logikai programozásban egy predikátum elvben kétféleképpen definiálható: tényállítások (hosszú) sorozatával – ez az *extenzionális* definíció, illetve klózokkal amit *intenzionális* definíciónak neveznek. Amikor egy Prolog program, azon belül egy predikátum tanulási feladatot összeállítunk, akkor rendszerint a tanulási példák halmaza tényállítások egy sorozata. Úgy is ki lehet fejezni a feladatot, hogy adott egy extenzionális definíció és keressük a vele leginkább megegyező intenzionális definíciót. A megtanulandó predikátum klózainak definíciójához felhasználhatunk háttértudást, amely a legkényelmesebben ugyancsak predikátum definíciók egy halmazaként kerülhet be rendszerbe. A háttértudás predikátumait azonban a rendszer tanulás közben nem változtathatja meg. Az alábbiakban formálisan is definiálásra kerül a logikai programok tanulása (ILP).

2.5. Definíció: θ helyettesítés, legáltalánosabb egyesítés [Lloy93]: A $\theta = \{X_1/t_1, X_2/t_2, \ldots X_k/t_k\}$ *helyettesítés* egy leképezés a változók és a termek között. A θ *helyettesítés* alkalmazása egy W formulára Wθ azt jelenti, hogy a W formulában előforduló összes X_j változót helyettesítünk a t_j termmel (1 j k). Legyen S formulák egy halmaza. A θ helyettesítést *egyesítésnek* nevezzük, ha Sθ egyetlen elemű. Egy adott S halmaz esetén, ha az S halmaz elemei egyesíthetők, akkor mindig létezik a *legáltalánosabb egyesítés*, amit a következőképpen lehet definiálni. A θ egyesítést *legáltalánosabb egyesítésnek* nevezik, ha tetszőleges σ egyesítésre létezik egy olyan τ helyettesítés, hogy $\sigma = \theta\tau$.

2.6. Definíció: Lefedés [Lav94]: adott B háttértudás, H hipotézis – mindkettő program klózok halmaza és e tanulási példa. A H hipotézis lefedi (vagy fedi) az e példát, ha az e a $B \cup H$ program logikai következménye. E példáknak egy halmaza. A H hipotézis lefedi E-t, akkor és csak akkor ha minden $e \in E$ példát lefed. A H hipotézis *teljes*, ha a pozitív példák halmazát lefedi (minden pozitív példát lefed). A H hipotézis *konzisztens*, ha a negatív példák halmazából egyetlen elemet sem fed le.

2.7. Definíció: A logikai programok tanulásának (ILP) alapfeladata [Lav94]: adott B háttértudás, H_0 kezdeti hipotézis, E^+ pozitív példák halmaza, E^- negatív példák halmaza. Az induktív logikai tanuló rendszer feladata egy teljes és konzisztens H hipotézis előállítása a H_0-ból kiindulva.

A részletekben való elmélyülés előtt, már csoportosíthatók a tanuló algoritmusok néhány szempont szerint. Egyes tanuló algoritmusok csak pozitív, mások csak negatív példákat tudnak feldolgozni. A gyakorlati életben sok az olyan feladat, amelyben csak pozitív példák fordulnak elő (pl. betegségek okai, balesetek körülményei, ásványkincsek lelőhelyei). Azok az esetek ismertek, hogy mikor járnak együtt egyes tünetek egy bizonyos

---

el lehet helyezni, ezután a fa megváltoztatható, majd a program kiegészíthető az új formulával (assert), a régi formulát ha szükséges ki lehet a programból törölni (retract).

betegséggel, de az nem, hogy mikor nem járnak együtt. Ugyanígy nem mondhatjuk meg, hogy mikor biztosan nem történik baleset, mikor biztosan nem található egy ásványkincs egy adott területen. Azokban az esetekben, amikor csak a pozitív példákat veszi figyelembe a tanuló algoritmus, akkor új klóz hozzáadásával, vagy ritkábban a meglévő program egy klózának általánosításával javítható a kezdeti hipotézis. Ebben a körben is előfordulhatnak azért negatív példák, amelyeket a rendszer az új klózok előállításakor vesz figyelembe, azaz csak olyan új klózt ad hozzá a kezdeti hipotézishez, amely nem fed le negatív példákat.

Van azonban egy kevésbé ismert módja a tanulásnak: amikor egy kiindulási hipotézist a negatív példák alapján módosítanak. Ekkor a tanuló algoritmus feladata a kezdeti hipotézis szűkítése a klózok specializálásával, esetleg törlésével. A pozitív példák a specializáció egy keretét biztosítják, azaz csak olyan specializációt engednek meg, amelynek a hatására az új hipotézis viselkedése a pozitív példákon nem romlik, továbbra is minden pozitív példát le fog fedni. A módszerhez alkalmazásához szükség van egy kiindulási hipotézisre. Ezt az ágát a tanulásnak nevezik *abduktív* tanulásnak. Az ilyen elveken alapuló rendszereket *elmélet revíziós* (theory revision) rendszereknek is nevezik.

További lehetséges felosztás az, hogy az algoritmus igénybe veszi-e egy tanár (*bölcs*) segítségét. Ha nem, akkor az algoritmus egyszerre megkaphatja az összes (pozitív és negatív) példát, majd azokon dolgozik. Az ilyen algoritmusokat *empirikus* tanuló algoritmusnak nevezik [Lav94]. *Interaktív* algoritmusok esetben egy *bölcs* egyenként adja meg a tanulási példákat, a pozitívokat és a negatívokat felváltva. A programnak egy példa feldolgozásával végeznie kell, mielőtt a *bölcs* új példát ad. Előnye az algoritmusnak, hogy az algoritmus a főbb elágazási pontokon kérhet javaslatokat a *bölcstől* a folytatás irányát illetően. Hátránya pedig az, hogy a *bölcs* esetleges hibáira fel kell készülnie, vissza kell tudni térnie egy korábbi állapotba, a *bölcs* elfárad ha túl sok kérdésre kell válaszolni, a kérdésekre esetleg hibásan válaszol stb. Máig megoldatlan kérdés az interaktív tanuló algoritmusok stabilitása. Ugyanazokat a hipotéziseket tanulja-e meg az algoritmus, ha a példákat más sorrendben kapja meg? Kevés minta esetén általában ez nem igaz. A természetes nyelvi [Ale97] és az egészségügyi alkalmazásoknál [KókP96], [Kók97] is előfordult, hogy egy kezdeti (általános) hipotézist kellett a negatív példák alapján specializálni. Ez azt mutatja, hogy az abduktív tanulásnak egyes alkalmazásokban fontosabb szerep jut, mint az indukciónak. A szerzőt ez is motiválta a 3. Fejezetben ismertetésre kerülő IMPUT algoritmus kifejlesztésében.

2.3. Induktív tanulási módszerek

Ebben a fejezetben néhány ILP tanuló algoritmus bemutatására kerül sor. A számos lehetséges ILP tanuló algoritmus közül azok kerültek kiválasztásra, amelyek valamilyen módon kapcsolódnak a dolgozat későbbi fejezeteihez.

Első lépésként tegyük fel, hogy már létezik egy eljárás arra, hogy megtanuljon egy szabályt a pozitív és a negatív példák figyelembevételével. Természetesen az eljárás által szolgáltatott szabály nem fogja az összes pozitív példát lefedni. Azt is tegyük fel, hogy az eredményként kapott szabály nem ütközik a negatív példákkal.[7] Hogyan építhetünk fel ebből egy olyan módszert, ami alkalmas az összes pozitív példát lefedő szabályrendszer létrehozására? A válasz a szekvenciális lefedési algoritmus (*sequential covering algorithm*).

2.3.1. A szekvenciális lefedési algoritmus

A 2.4. ábrán ismertetésre kerülő szekvenciális lefedési algoritmus számos tanuló algoritmus fő vázát alkotja. Az algoritmus egy mohó keresést valósít meg visszalépés nélkül, ezért az eredményt illetően semmit sem lehet állítani. Nem állítható, hogy a legrövidebb és az sem hogy a legjobb szabályokat eredményezi. Egymástól független *vagy* kapcsolatban álló (diszjunktív) szabályrendszert hoz létre. Ha a végén megmaradó kevés számú pozitív példát már nem vagy csak kis jóságú szabályokkal tudja lefedni, akkor az algoritmus megáll.

A következőkben azt vizsgáljuk meg, hogy milyennek kell lennie annak az algoritmusnak, amely képes megtanulni egy szabályt a példák alapján. Természetesnek tűnik, hogy a szabály pontossága a lehető legnagyobb kell legyen, viszont nincs szüksége arra, hogy minden példát egyszerre le tudjon fedni.

---

[7] Két eset lehetséges: a tanuló eljárás figyelembe tudja venni a negatív példákat, vagy pedig a tanuló rendszer nem engedi meg a negatív példákat.

Tekintettel arra, hogy a továbbiakban Prolog program klózok tanulásáról lesz szó, természetesen a szabályok megtanulása egy klóz meghatározását jelenti. A klóz feje ismert, ott a megtanulandó predikátum áll, az aritásának megfelelő számú különböző változóval. A feladat a klóz törzsében szereplő literáloknak a meghatározása, továbbá a számos új változó közötti kapcsolat biztosítása. A klóz törzsében a háttér tudásban szereplő predikátumok lehetnek, illetve rekurzív szabály esetén maga a megtanulandó predikátum. Ha a tanuló algoritmus nem tudja szimbolikusan kezelni a változók közti kapcsolatot, akkor a törzsben utólag beszúrt A=B, vagy ¬A=B alakú literálok is előfordulhatnak.[8]

```
SZEKVENCIÁLIS_LEFEDÉS(Attribútumok, Példák, Küszöbérték)
begin
  Szabályok  = {}
  új_szabály = EGY_SZABÁLY_TANULÁSA(Attribútumok, Példák)
  while TELJESÍTMÉNY(új_szabály, Példák) > Küszöbérték do
  begin
    Szabályok = Szabályok + új_szabály
    Példák = Példák - az új_szabály által lefedett példák
    új_szabály = EGY_SZABÁLY_TANULÁSA(Attribútumok, Példák)
  end while
  A Szabályok halmaz rendezése a TELJESÍTMÉNY értékek alapján
end SZEKVENCIÁLIS_LEFEDÉS
```

2.4. Ábra. A szekvenciális lefedési algoritmus

Amikor klóz törzséhez egyre újabb literálokat adnak, az tekinthető úgy, hogy a klózt specializálják. Egyre több feltételnek kell teljesülni ahhoz, hogy a klóz feje igaz legyen. Ezt a stratégiát úgy nevezik, hogy az *általánostól a speciális felé* haladás. Természetesen lehetséges a fordított irány is, amikor a kezdetben felsorolt literálokból elhagynak, például termek helyett új változó bevezetésével, aminek köszönhetően bizonyos literálok egybeesnek. Ez a stratégia a *speciálistól az általános* felé haladás.

A tanuló algoritmusokban között mind a kettő előfordul, például a CN2 [Cla89] tanuló algoritmus amely, az ID3 egy átfogalmazása egy döntési fával ekvivalens elsőrendű formula halmazt tud megtanulni, olyan program klózokat, amelynek a jobb oldalán csak A=*t* literálok vannak (A egy változó, *t* pedig egy term). A klózokat az *általánostól a speciálisig* stratégiával határozta meg. A „mohó" szekvenciális lefedés algoritmus helyett azonban egy fejlettebb ún. *k*-sugár keresés (*k*-beam search) módszert építettek bele. Ennek a keresési módszernek az a lényege, hogy egy *k* elemű memóriában a mindenkori *k* db legjobb részeredményt (a klóz jobb oldalára szánt literál sorozatot) tárolja. Újabb literál hozzáadásakor a memóriában lévő összes formulához hozzáilleszti az új literált, ezekből ismét kiválasztja a legjobb *k* elemű legjobbat és így tovább. Amennyiben a *k=1*, akkor az eredeti szekvenciális lefedési módszert kapjuk. A *speciálistól az általános* felé haladó szabály kialakítási stratégiát alkalmaz például a GOLEM [Mugg90] tanuló algoritmus.

2.8. Definíció: θ-magába foglalás (θ-subsumption) [Lav94]: A c klózt θ-magába foglalja a c' klóz, ha létezik egy olyan θ helyettesítés, hogy a $c\theta \subseteq c'$, azaz a θ helyettesítés után az eredményül kapott klóz minden literálja szerepelni fog a c' klózban. A klózokat a 2.4. definíció e. pontja szerint literálok halmazának tekintjük. Például a

```
c: lánya(X, Y) ← szülője(Y, X).
```

klózt θ-magába foglalja az alábbi c' klóz, a $\theta = \varnothing$ helyettesítéssel

```
c': lánya(X, Y) ← szülője(Y, X), nő(Y).
```

Ez a definíció bevezet a klózokon egy rendezést, ami az *általánossággal* függ össze. Definíció szerint a c klóz legalább olyan általános mint a c' klóz ($c \leq c'$), ha a c klózt θ-magába foglalja a c'. A c klóz általánosabb, mint a c' ($c < c'$), ha a $c \leq c'$ teljesül, de a $c' \leq c$ nem teljesül. Ekkor a c a c' klóz *általánosítása*, a c' klóz a c klóz *specializációja* (*speciális esete*). Amennyiben $c \leq c'$, akkor $c \rightarrow c'$ is teljesül, a $c \rightarrow c'$ teljesüléséből viszont nem következik, hogy $c \leq c'$ [Plot71].

[8] Az „=" a Prologban az egyesítés (unification) operátor.

A fenti definíciónak megfelelően a klóz jobb oldalához történő literál hozzáadás egy *specializáció*. Az *általánostól a speciálisig* stratégia során az elsőrendű formulák szintaxisa kijelöli az adott klózból θ helyettesítéssel kapható speciálisabb klózokat (nemcsak literál hozzáadás lehetséges, hanem más specializációs lehetőség is van). Ebben a keresési térben kell megtalálni azokat a formulákat, amelyek lehetséges hipotézisei lesznek a tanuló algoritmusnak. A literál hozzáadás egy különösen kritikus lépés. Ekkor ugyanis a keresési fa hirtelen kiszélesedik amiatt, hogy a klózok jobb oldalán elvben tetszőleges számú, azonos vagy különböző predikátumot tartalmazó literál állhat. A gyakorlati feladatok során célszerűnek mutatkozott beépíteni a tanuló algoritmusokba egy lehetőséget arra, hogy korlátozásokat lehessen megadni a jobb oldalon álló literálok számára és predikátumára. Az ilyen megszorításokat *nyelvi megszorításnak* (language bias) nevezik.

2.3.2. A FOIL tanuló algoritmus

Ha a C 4.5 vagy a CN2 tanuló algoritmussal megkíséreljük megtanulni a rokonsági kapcsolatokat, például a `lánya/2` relációt, akkor az a következőképpen nézne ki. Mivel két személy szerepel a relációban, ezért mindkettő neve, apja neve, anyja neve, férfi, nő adatai szükségesek, ez 2 x 5 = 10 attribútum. A tréning adatok egy olyan fájlt alkotnak, amelyben egy sor egy példát reprezentál, a soron belül pedig 11 adat található (10 bemenő adat, valamint a predikátum értéke). A megtanult propozíciók pedig ehhez hasonlók lesznek:

```
if (apja_neve1 = Ferenc) ∧ (nő1 = igaz) ∧ (neve2 = Ferenc) then lánya = igaz
if (apja_neve1 = Lajos)  ∧ (nő1 = igaz) ∧ (neve2 = Lajos)  then lánya = igaz
...
```

Ha növeljük a tréning adatok számát, a megtanult formulák száma egyre nőni fog, egyre kevésbe áttekinthető szabályrendszert kaphatunk, amely természetesen elvileg egyre pontosabban leírja a kívánt relációt. A propozícionális rendszerek azonban sohasem lesznek képesek egyetlen zárt formulában kifejezni a megoldást. Elsőrendű tanuló programok esetén azt reméljük, hogy a megtanult szabály a következő lesz:

$$\forall X \forall Y \; \text{apja}(Y, X), \; \text{nő}(Y) \rightarrow \text{lánya}(X, Y).$$

Ez világosan mutatja, hogy mennyivel nagyobb a használati értéke egy elsőrendű tanuló algoritmusnak. A megtanult formulák lehetnek rekurzívak, lehetnek bennük változók (akár olyanok is, amelyek csak a jobb oldalon fordulnak elő), függvényszimbólumok, fák, listák. A legelső elsőrendű tanuló algoritmusok egyike volt a FOIL [Quin90], amely az ID3-at is kifejlesztő Quinlan és munkatársai nevéhez fűződik.[9]

```
FOIL(Cél_Predikátum, Háttér_Predikátum, Példák)
begin
  Poz = a Példák közül a pozitív példák
  Neg = a Példák közül a negatív példák
  Tanult_szabályok = {}
  while Poz nem üres halmaz do
  begin  /* egy szabály tanulása  */
    Új_Szabály = a fej: a Cél_predikátum, minden változó új, a jobb oldal: üres.
    ÚjSzabNeg = Neg  /* az új szabály által eddig lefedett negativ példák */
    while ÚjSzabNeg nem üres halmaz do
    begin /* specializálás, amíg a lefedett negatív példák halmaza üres nem lesz */
      PossLiterals = generálja az összes szóba jöhető literált
      BestLiteral = kiválasztja a legjobbat a Foil_Gain függvény értéke alapján
      Új_Szabály = Új_Szabály + BestLiteral (a literál hozzáadása a jobb oldalhoz)
      ÚjSzabNeg = az Új_Szabály által még lefedett példák (a többi törölhető)
    end  while ÚjSzabNeg nem üres
    Tanult_szabályok = Tanult_szabályok + Új_Szabály
    Poz = Poz - az Új_Szabály által lefedett pozitív példák
  end  while Poz nem üres
end  FOIL
```

2.5. Ábra A FOIL algoritmus vázlata

A következőkben röviden bemutatásra kerül a FOIL algoritmus [Mit97] alapján. A FOIL pozitív és negatív példák sorozatát kapja meg inputként. Amennyiben van, úgy háttértudás is megadható extenzionális formában,

---

[9] A FOIL rendszer legutóbbi 6.4-es verziója ingyenesen letölthető a ftp://ftp.cs.su.oz.au/pub/foil6.sh Internet címről.

azaz tények sorozataként.[10] Az algoritmus a szekvenciális lefedés módszerét használja valamint az *általánostól a speciálisig* stratégiát. Az alábbiakban bemutatásra kerül az algoritmus vázlata.

Az algoritmus két egymásba ágyazott ciklust tartalmaz. A külső ciklus a szekvenciális lefedés egy megvalósítása, míg a belső ciklus a specializáció folyamatát vezérli. A FOIL egy egyszerű „mohó" kereséssel indul el a legjobbnak mutatkozó literál irányába. A klózok kezdetben üres jobb oldalához egyre újabb és újabb literálokat illeszt. Ezután egy *Foil_Gain* mérték alapján kiválasztja a legjobb literált. Ezt hozzáilleszti az eddigi klózhoz, majd újabb literálokat generál, ismét kiválasztja a legjobbat stb. A folyamat akkor ér véget, amikor a kapott specializált klóz már egyetlen negatív példát sem fed el. Tegyük fel, hogy van már egy

$$P(X_1, X_2, \ldots X_k) \leftarrow L_1, L_2, \ldots, L_n.$$

alakú klóz, ahol $P(X_1, X_2, \ldots X_k)$ a klóz feje, az L_1, L_2, ..., L_n pedig az eddig literálok. A FOIL a következő lépésben egy újabb L_{n+1} literált illeszt az eddigi jobb oldalhoz, amely a következő alakú lehet:

- $Q(V_1, V_2, \ldots, V_m)$, ahol a Q egy ismert predikátum (vagy egyezik a megtanulandó predikátummal, vagy benne van a háttér tudásban). A V_1, V_2, ..., V_m változók vagy teljesen újak, vagy már szerepelnek a klózban. Legalább egy változónak azonban olyannak kell lennie, amely már korábban előfordult a szabályban!
- $\neg Q(V_1, V_2, \ldots, V_m)$, ahol a Q egy ismert predikátum (vagy egyezik a megtanulandó predikátummal, vagy benne van a háttér tudásban). A V_1, V_2, ..., V_m változók vagy teljesen újak, vagy már szerepelnek a klózban. Legalább egy változónak azonban olyannak kell lennie, amely már korábban előfordult a szabályban!
- $X_s = X_t$, ahol X_s és az X_t változók már előfordultak a szabályban
- $\neg X_s = X_t$, ahol X_s és az X_t változók már előfordultak a szabályban

Egy rövid példán ezt a következőképpen lehet bemutatni. Tegyük fel, hogy a megtanulandó predikátum a `lány_unoka/2`. A kiindulási klóz természetesen a `lány_unoka(X, Y) ←`. Az első lépésben a FOIL a következő lehetséges literálokat generálja: `X=Y`, `nő(X)`, `nő(Y)`, `apja(X,Y)`, `apja(Y,X)`, `apja(X,Z)`, `apja(Z,X)`, `apja(Y,Z)`, `apja(Z,Y)`, `¬X=Y`, `¬nő(X)`, `¬nő(Y)`, `¬apja(X,Y)`, `¬apja(Y,X)`, `¬apja(X,Z)`, `¬apja(Z,X)`, `¬apja(Y,Z)`, `¬apja(Z,Y)`. A Z egy új változó, amely még nem fordult elő a szabályban, azaz a Z nem azonos sem az X-szel, sem az Y-nal. Tegyük fel, hogy a legjobbnak az `apja(Y,Z)` literál bizonyul. Ekkor a szabály a következő lesz:

```
lány_unoka(X, Y) ← apja(Y, Z).
```

A következő lépésben a FOIL lehetséges literálként az előző lépésben megadottakat adja meg, valamint néhány újabbat is: `nő(Z)`, `Z=X`, `Z=Y`, `apja(Z,W)`, `apja(W,Z)`, `¬nő(Z)`, `¬Z=X`, `¬Z=Y`, `¬apja(Z,W)`, `¬apja(W,Z)`. A W egy eddig elő nem fordult új változó. Ha most a FOIL az `apja(Z,X)` literált választja, majd a harmadik lépésben a `nő(Y)`-t, akkor a következő szabályt kapjuk:

```
lány_unoka(X, Y) ← apja(Y, Z), apja(Z,X), nő(Y).
```

A FOIL a legjobb literál kiválasztásához egy nyereségfüggvényt használ. A *Foil_Gain* függvénynek két változója van, az L az új literál, az R az eddig már meghatározott szabály. Az R' szabály úgy kapható meg, hogy az R szabályhoz hozzáillesztjük az L literált. Az R és az R' szabályokban szereplő változókat a FOIL az összes lehetséges módon leköti, egy lekötést pozitívnak neveznek, ha a klóz feje – amely a lekötés után már változó mentés tény lesz – a pozitív példák között explicit módon előfordul. Az összes többi esetben a lekötés negatív (ha a tény a negatív példák között fordul elő, vagy ha sem a pozitív sem a negatív példák között nem fordul elő). Az R szabály szerinti pozitív lekötések száma p_0, a negatív lekötések száma n_0, az R' szabály pozitív lekötéseinek száma p_1, a negatívoké n_1. A t paraméter az R szabály olyan pozitív lekötéseinek a száma, amelyek az R' szabályban is pozitív lekötések maradnak. Ha az R' szabályban új változó jelenik meg, akkor azt az összehasonlításban nem veszik figyelembe.

$$Foil\_Gain\ (L,R) = t\left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0}\right)$$

[10] A FOIL algoritmus ANSI C-ben készült, és nem tartalmaz program klózok beolvasására, kiértékelésére alkalmas alrendszert. A formulákat C adatstruktúrákkal reprezentálták.

A fentebb ismertetett FOIL algoritmus képes rekurzív szabályokat is meghatározni. Ezt egyedül a fent említett *Foil_Gain* nyereség befolyásolja. Fontos megjegyezni, hogy a FOIL algoritmus csak zajmentes adatokon működik jól. Ahhoz, hogy zajos adatokon is működő gyakorlati rendszer lehessen, különböző védekező mechanizmusokat építettek a rendszerbe pl. a jobb oldalak hosszára vonatkozó korlátot, vagy az egy szabályban a használt változók számára egy felső korlátot adtak meg. A FOIL-t kiegészítették egy utólagos egyszerűsítő algoritmussal is.

2.3.3. A Chillin tanuló algoritmus

A Chillin tanuló algoritmus Zelle és Mooney nevéhez fűződik.[11] Az alábbi ismertetés a [Zel94] publikáció alapján készült. A Texas egyetem kutatói egy kombinált szabály keresési stratégiát valósítottak meg, amely az *általánostól a speciálisig* és a *speciálistól az általánosig* – az ő szóhasználatukkal élve felülről-lefelé (top-down) és alulról-felfelé (bottom-up) – módszert is használ a tanulás során. A Chillin tekinthető több már meglévő rendszer hasznos kombinációjának. Az alulról-felfelé (speciálistól az általános felé) stratégia alapvetően a GOLEM [Mugg90], a felülről-lefelé stratégia pedig a FOIL [Quin90] tanuló algoritmus egy jelentősebb továbbfejlesztése. Mind a GOLEM, mind pedig a FOIL tanuló algoritmus csak extenzionális háttértudást tudott felhasználni. Egyikben sem volt lehetőség függvényszimbólumok használatára. A Chillin rendszer kiküszöbölte a nagy méretű adatállományok használatából eredő problémákat és megengedi a példák valamint a háttértudás megadását szabályos Prolog klózok formájában. Ez nagymértékben annak köszönhető, hogy a rendszert Quintus Prolog programozási nyelven implementálták.

A Chillin tanuló algoritmus a természetes nyelvi alkalmazásai miatt szerepel a dolgozatban. Készítői egy természetes nyelvi interfész szintaktikus szabályainak tanulására használták. A természetes nyelvi interfészek egy más módon történő előállításával a 4.3 Fejezet foglalkozik majd. A Chillin rendszer a tanuló algoritmusok között is említésre méltó, több érdekes tulajdonsággal rendelkezik: pl. új predikátumot hoz létre szükség esetén.

A tanuló algoritmus lényegében egy kompaktáló algoritmus, amely a pozitív példák nagy halmazából kiindulva egyre kisebb programokat állít elő, amely lefedi a pozitív példákat. A legspeciálisabb tényekből indul el, majd általánosításokat vezet be, amelyek következtében a kezdeti program kompaktsága csökken. A kompaktságot a CIGOL[12] [Mugg88] tanuló algoritmusnál használt képlet alapján számítja ki. A kompaktság egyszerűen a szintaktikus mérete a programnak. A lehetséges hipotézisek közül a „mohó" algoritmus egyszerűen a legjobb kompaktságot elérőt választja. Az algoritmus külső váza egy alulról-felfelé stratégiának felel meg:

```
DEF = { E ←. | E ∈ Poz}  /* a pozitív példákból klózok készítése,
                Amelyekben a jobb oldalak üresek */
repeat
  PAIRS = véletlenszerűen kiválasztott k db pár a DEF halmazból
  GENS = {G | G = Build_gen(<Ci, Cj>, DEF, Poz, Neg)}
         /* minden <Ci, Cj> ∈ PAIRS klóz párra egy G általánosítás kiszámítása */
  Best = a GENS halmazban lévő legnagyobb kompaktságot biztosító klóz
  DEF = (DEF - a Best által θ-magában foglalt klózok) + Best
until nincs további kompaktálási lehetőség
```

2.6. Ábra A Chillin algoritmus külső váza

Az algoritmus véletlenszerűen kiválasztott klóz párokból indul ki, amelyekre a Plotkin által bevezetett *lgg* (Least General Generalization) [Plot70] *legkisebb mértékű általánosítás* műveletet alkalmazza. A szabály végső formájának kialakításához még korrigálja a klózok jobb oldalát a negatív példáknak megfelelően. Természetesen az egyes pozitív példa párokból kiindulva különböző általánosításokat kaphat, amelyek közül a legjobbat fogja kiválasztani. Az új klóz által lefedett pozitív példákat, sőt esetleg korábban bevezetett, az új klóz által magába foglalt speciálisabb klózokat is eltávolítja a DEF-ből. A következő lépésben ismét véletlenszerűen választ klóz párokat, azokat általánosítja, kiválasztja a legjobbat, és így tovább, amíg csak van lehetséges összevonás.

---

[11] A Chillin tanuló rendszer ingyenesen letölthető a ftp://ftp.cs.utexas.edu/pub/mooney/chillin/ Internet címről. A név a CHILL INduction kifejezés rövidítése, amely utal arra, hogy a CHILL természetes nyelvi rendszerhez készített szintaxis tanuló programról van szó.

[12] A CIGOL visszafelé olvasva LOGIC [Mugg95].

2.9. Definíció: *lgg* [Lav94]: c_1 és c_2 klózok, akkor $lgg(c_1, c_2)$ a θ-magában foglalás háló szerinti legkisebb felső korlátja c_1-nek és c_2-nek. Az *lgg* kiszámítása az alábbiak szerint történik

termekre:
a.) $lgg(t, t) = t$
b.) $lgg(f(s_1, s_2, ..., s_n), f(t_1, t_2, ..., t_n)) = f(lgg(s_1, t_1), lgg(s_2, t_2), ..., lgg(s_n, t_n))$
c.) $lgg(f(s_1, s_2, ..., s_m), g(t_1, t_2, ..., t_n)) = V$, ahol $f \neq g$, V pedig egy változó, amely reprezentálja az $lgg(f(s_1, s_2, ..., s_m), g(t_1, t_2, ..., t_n))$ értékét
d.) $lgg(s, t) = V$, ahol $s \neq t$ és legalább az egyikük változó, akkor V reprezentálja az $lgg(s, t)$ értékét
atomokra:
e.) $lgg(p(s_1, s_2, ..., s_n), p(t_1, t_2, ..., t_n)) = p(lgg(s_1, t_1), lgg(s_2, t_2), ..., lgg(s_n, t_n))$
f.) $lgg(p(s_1, s_2, ..., s_m), q(t_1, t_2, ..., t_n))$ nem definiált, ha $p \neq q$
literálokra:
g.) $lgg(L_1, L_2)$ = az e.) és az f.) pontok szerint ha L_1 és L_2 pozitív literál
h.) $lgg(\neg L_1, \neg L_2) = \neg lgg(L_1, L_2)$
i.) $lgg(L_1, L_2)$ = nem definiált, ha L_1 és L_2 közül az egyik pozitív a másik pedig negatív literál
klózokra:
j.) $c_1 = \{L_1, ..., L_n\}$, $c_2 = \{K_1, ..., K_m\}$ akkor
$lgg(c_1, c_2) = \{N_{ij} \mid N_{ij} = lgg(L_i, K_j), L_i \in c_1, K_j \in c_2$ és az $lgg(L_i, K_j)$ értelmezett$\}$

Néhány példa a fenti szabályok alkalmazására:

$lgg([a, b, c], [a, c, d]) = [a, X, Y]$
$lgg(f(a, a), f(b, b)) = f(lgg(a, b), lgg(a, b)) = f(V, V)$. Fontos megjegyezni, hogy az azonos termek általánosításához azonos változót kell hozzárendelni. Ez érvényes atomokra, literálokra és klózokra is!
lgg(szülő(*Anna*, *Mária*), szülő(*Anna*, *Tamás*)) = szülő(*Anna*, X)
lgg(szülő(*Anna*, *Mária*), ¬szülő(*Anna*, *Tamás*)) = nem definiált
ha c_1: lánya(*Mária*, *Anna*) ← nő(*Mária*), szülője(*Anna*, *Mária*), és
c_2: lánya(*Éva*, *Tamás*) ← nő(*Éva*), szülője(*Tamás*, *Éva*), akkor az $lgg(c_1, c_2)$ =
lánya(X, Y) ← nő(X), szülője(Y, X), ahol X az *lgg*(*Mária*, *Éva*) és Y = *lgg*(*Anna*, *Tamás*) helyett áll.

2.10. Definíció: *rlgg* (Relative Least General Generalization) [Plot70]: e_1 és e_2 tények (változómentes atomok – példák), B háttértudás, akkor $rlgg(e_1, e_2) = lgg(e_1 \leftarrow K, e_2 \leftarrow K)$, ahol K (az extenzionális) háttértudásban szereplő összes tény konjukciója.

Például a fenti példa adatait felhasználva, ha e_1: lánya(*Mária*, *Anna*), e_2: lánya(*Éva*, *Tamás*), a háttértudásban szereplő tények konjunkciója K: szülő(*Anna*, *Mária*), szülő(*Anna*, *Tamás*), szülő(*Tamás*, *Éva*), szülő(*Tamás*, *Ian*), nő(*Anna*), nő(*Mária*), nő(*Éva*). Az itt felsorolt paraméterekkel a $rlgg(e_1, e_2) = lgg(e_1 \leftarrow K, e_2 \leftarrow K)$ lesz, pontosabban:

lánya($V_{MÁRIA,ÉVA}$, $V_{ANNA,TAMÁS}$) ← szülő(*Anna*, *Mária*), szülő(*Anna*, *Tamás*), szülő(*Tamás*, *Éva*),
szülő(*Tamás*, *Ian*), nő(*Anna*), nő(*Mária*), nő(*Éva*), szülő(*Anna*, $V_{MÁRIA,TAMÁS}$),
szülő($V_{ANNA,TAMÁS}$, $V_{MÁRIA,ÉVA}$), szülő($V_{ANNA,TAMÁS}$, $V_{MÁRIA,IAN}$),szülő($V_{ANNA,TAMÁS}$, $V_{MÁRIA,IAN}$),
szülő($V_{ANNA,TAMÁS}$, $V_{TAMÁS,ÉVA}$), szülő($V_{ANNA,TAMÁS}$, $V_{TAMÁS,IAN}$), szülő(*Tamás*, $V_{ÉVA,IAN}$),
nő($V_{ANNA,MÁRIA}$), nő($V_{ANNA,ÉVA}$), nő($V_{MÁRIA,ÉVA}$).

ahol $V_{L,M}$ jelöli az *lgg*(L,M) helyére bevezetett új változót minden esetben. Amennyiben egyszerűsítjük az így kapott klózt, és a nem releváns literálokat töröljük, akkor egy már ismerős eredményhez juthatunk:

lánya($V_{MÁRIA,ÉVA}$, $V_{ANNA,TAMÁS}$) ← szülő($V_{ANNA,TAMÁS}$, $V_{MÁRIA,ÉVA}$), nő($V_{MÁRIA,ÉVA}$).

ha a változókat helyettesítjük X és Y-nal, akkor

lánya(X, Y) ← szülő(Y, X), nő(X).

A definíciók után a 2.7. ábrán a Chillinben alkalmazott Build_gen algoritmus kerül bemutatásra, amely két véletlenszerűen kiválasztott pozitív példát általánosít, majd a negatív példákat figyelembe véve specializál, amennyiben szükséges:

A literál hozzáadás teljes mértékben a FOIL algoritmus szerint történik: a Chillin is a lehetséges literálok egy halmazát generálja, az egyes literálokat a klózok jobb oldalára próbálja, majd egy nyereségfüggvényt számít ki. Ez a nyereségfüggvény a *Foil_Gain* egy módosítása. A legjobb literállal dolgozik tovább és további literálokat kapcsol a klóz jobb oldalához, amíg ez lehetséges, vagy a lefedett negatív példák elfogynak. A literálok hozzáadása közben előre nem látott akadály is felléphet, nevezetesen, ha nem talál az algoritmus olyat,

amellyel további nyereséget lehetne elérni, valamint vannak még a klóz által lefedett negatív példák. Ez a szituáció akkor fordul elő, ha nem áll rendelkezésre a háttértudásban olyan predikátum, amellyel szét lehetne választani a pozitív és a negatív példákat. Erre az esetre is tartalmaz egy megoldást a Chillin, új predikátum bevezetését. Ennek az alapötlete a CHAMP [Kij92] rendszerből származik. A rendszer az egyes változók lehetséges lekötéseit vizsgálva olyan partícionálásokat keres, amelyek szétválasztják az adott klóz által lefedett pozitív és a negatív példákat (amelyeket nem lenne szabad lefednie a klóznak). Ha ilyen partíció van, akkor erre egy új predikátumot generál. Az új predikátum segítségével a rendszer már egy *konzisztens* klózt tud előállítani.

```
Build_gen((<Ci, Cj>, DEF, Poz, Neg)
begin
  GEN = lgg(Ci, Cj), a 2.9. definíció szerint
  CNEGS = a GEN által lefedett negatív példák halmaza
  if CNEGS = üres then return GEN

  modGEN = add_literals(Poz, CNEGS, GEN)
  CNEGS = a modGEN által lefedett negatív példák halmaza
  if CNEGS = üres then return modGEN

  RedukáltDEF = DEF - a modGEN által θ-magába foglalt klózok
  Cpoz = olyan pozitív példák, amelyeket a RedukáltDEF nem fed le
  Literal = új_predikátum(CPoz, CNEGS, modGEN)
  modGEN = modGEN + Literal
  return modGEN
end
```

2.7. Ábra A Chillinben alkalmazott Build_gen eljárás vázlata

2.3.4. A Progol tanuló algoritmus

A Progol[13] [Mugg95] tanuló algoritmust Muggleton és munkatársai fejlesztették ki a Yorki Egyetemen. A rendszer a korábbi CIGOL [Mugg88], és a GOLEM [Mugg90] tanuló algoritmusok egy továbbfejlesztése. Az előzőkhöz hasonlóan a pozitív példák közül véletlenszerűen kiválasztott párok általánosításával tanul újabb klózokat. Míg azonban az indukcióra a CIGOL az itt nem ismertetett *inverz rezolúciót*, a GOLEM rendszer az 2.10. definíció szerinti *rlgg* transzformációt, továbbá a Chillin a Plotkin által bevezetett *lgg* transzformációt [Plot70] alkalmazta, addig a Progolban az *inverz logikai következmény* (*inverse entailment*) transzformációt használták.

A 2.3.1. Fejezet végén már szóba kerültek a nyelvi megszorítások. A nyelvi megszorítások segítségével szűkíthető az a keresési tér amelyben a tanuló algoritmus mozog, amikor egy újabb hipotézist keres. Az elsőrendű tanulásnál a nyelvi megszorítás a megtanulandó klózokra vonatkozik, azoknak is a jobb oldalára. A Progol rendszer készítésekor létrehoztak egy a nyelvi feltétel leírását lehetővé tevő reprezentációt, amelyet *definit mód nyelvnek* neveztek. Ez egy továbbfejlesztése a logikai programozási nyelvekben alkalmazott típus leírásnak.

2.11. Definíció: Definit mód nyelv (Definite Mode Language) [Mugg95]: Legyen C egy definit klóz, amelynek a literáljain egy teljes rendezés van értelmezve, M mód deklarációk egy halmaza. A C: $h \leftarrow b_1, b_2, ..., b_n$ definit klóz benne van az M deklarációk által megengedett $\mathcal{L}$(M) mód nyelvben, amennyiben:

1. a h klóz fejre létezik egy `modeh` (mode head) deklaráció az M halmazban. A deklarációban előforduló `+típus`, `-típus` szimbólumok helyén *változók*, a `#típus` szimbólumok helyén *változómentes termek* állnak. A + jel a bemenő (input) változókat, a - a kimenő (output) változókat, a # pedig a konstans bemenő paramétereket jelöli.
2. Minden b_i literálra a klóz törzsében létezik `modeb` (mode body) deklaráció az M halmazban. A deklarációban előforduló `+típus`, `-típus` szimbólumok helyén *változók*, a `#típus` szimbólumok helyén *változómentes termek* állnak.
3. Minden `+típus` helyén álló változóra a b_i literálban teljesül, hogy: vagy előfordul `+típus` változóként h-ban, vagy `-típus` változóként egy b_j 1 j <i literálban.

---

[13] A Progol tanuló programnak az ANSI C verziója ingyen letölthető a ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/progol4.4/ Internet címről, a SICStus Prolog verzió pedig ftp://ftp.cs.york.ac.uk/pub/ML_GROUP/Papers/sicstus.pprogol271.tar.gz Internet címről.

2.12. Definíció: Mélységében korlátozott definit mód nyelv (Depth-bounded Definite Mode Language) [Mugg95]: Legyen C egy definit klóz, amelynek a literáljain egy teljes rendezés van értelmezve, M mód deklarációk egy halmaza. C benne van az M deklarációk által megengedett $\mathcal{L}_i(M)$ mód nyelvben, amennyiben benne van az $\mathcal{L}(M)$ mód nyelvben és minden változó mélysége legfeljebb *i*.

2.13. Definíció: Változó mélysége [Mugg95]:

1. A klóz fejében található változók mélysége definíció szerint *0*, $d(V) = 0$.
2. A klóz törzsében található változó mélysége a következő:

$$d(V) = \max_{u \in U_V} d(u) + 1$$

U_V azokat a (V-től különböző) változókat jelöli, amelyek a V változót tartalmazó literálokban vannak és amelyeknek a mélysége már ismert.

Például egy a faktoriális kiszámítására alkalmas klózokat tartalmazó *mód nyelv* megadható az alábbi M mód deklarációkkal:

```
modeh(*, f(+int, -int)).
modeb(*, d(+int, -int)).
modeb(*, f(+int, -int)).
modeb(*, m(+int, +int, -int)).
```

ahol, az `f` a faktoriális, a `d` dekrementálás (csökkentés), `m` pedig a szorzás predikátumát jelöli. Mindegyik két aritású, az első pozíción a bemenő paraméter, a másodikon a kimenő, a szorzás esetén az első kettő bemenő paraméter és a harmadik a kimenő. A * karakter az illető literálok maximális számát jelöli, a * egy elegendően nagy számot jelöl. Ennek a deklarációnak eleget tesz például az

```
f(A,B) ← d(A,C), f(C, D), m(A, D, B).
```

A fenti klózban a változók legnagyobb mélysége 2, ezért az $\mathcal{L}_i(M)$ nyelvnek csak akkor eleme, ha i 2. Pontosabban az *A* és *B* változók mélysége 0, a *C* változó mélysége 1, a *D* változó mélysége pedig 2.

Ezek után a Progol algoritmus a következőképpen formalizálható:

```
Progol(Példák, Háttér tudás)
begin
  Hipotézis := Háttér tudás
  while  a pozitív Példák halmaza nem üres do
    legyen e = egy változómentes tény a példák halmazából
    konstruáljunk meg ⊥i-t az e példából kiindulva (szaturáció)
    legyen a D klóz a ⊥i egy eleme, majd állítsuk elő a D általánosításait úgy,
       hogy literálokat hagyunk el a D klóz törzséből (redukció)
    legyen C klóz, a legnagyobb pontszámot elérő általánosítása a D klóznak
    Hipotézis := Hipotézis + C
    Példák := Példák - Lefedett(Példák, Hipotézis)  (cover removal)
  end  while
  return Hipotézis
end  Progol
```

2.8. Ábra A Progol algoritmus vázlata

A mélységben korlátozott mód nyelvtan lehetőséget biztosít arra, hogy egy ugyancsak mélységben korlátozott metainterpreter segítségével, egy rögzített *i* esetén, egy adott tanulási példát is figyelembe véve elő lehessen állítani az $\mathcal{L}_i(M)$ halmaz elemeit. A mód nyelvtan által generált klózok általában változómentes klózok, viszonylag sok literállal, ezt a lépést nevezik szaturációnak. A következő lépésben egy keresés következik, amely során a kiindulási klóz különböző általánosításait generálják úgy, hogy a klóz törzséből literálokat hagynak el, majd a kapott klózokhoz egy pontszámot rendelnek (redukció). Ebben a lépésben tulajdonképpen a θ-magában foglalás háló bejárása történik. Az általánosítással kapott klózt θ-magában foglalja a mód nyelvtannal generált klóz valamilyen θ helyettesítésre. Az algoritmus a legnagyobb pontszámot elért klózt hozzáadja a már rendelkezésre álló hipotézishez, majd végül a módosított hipotézis által lefedett pozitív példákat eltávolítja a példák halmazából (cover removal). Ezeket a lépéseket addig kell ismételni, míg a le nem fedett példák halmaza ki nem ürül.[14] Az a módszer, hogy literálok egy halmazából kell egyeseket elhagyni,

[14] Progol Felhasználói Kézikönyv: http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PProgol/ppman.html

ígéretesebbnek látszik, mint a FOIL-nál alkalmazott literál hozzáadás, amely esetben a keresési tér felülről nincs korlátozva [Mugg95].

2.3.5. A SPECTRE tanuló algoritmus

A SPECTRE[15] [Bos94] tanuló algoritmus kifejlesztés a svéd Boström nevéhez fűződik. Ez a tanuló algoritmus jelentős mértékben különbözik az eddigiekben látott algoritmusoktól. Ez ugyanis egy *abduktív* tanuló algoritmus, amely egy kezdeti hipotézis átalakításával próbál meg *konzisztens* végeredményhez eljutni. Az algoritmus számára megengedett műveletek a klóz törlés és az *unfolding* transzformáció, új klózt az algoritmus nem ad hozzá a kezdeti hipotézishez. Az algoritmus feltételezi azt is, hogy a kezdeti hipotézis induláskor minden pozitív példát lefed. A tanuló program feladata az, hogy a hipotézist úgy specializálja, hogy az egyre kevesebb negatív példát fedjen le. A tanuló algoritmus akkor áll meg, ha a hipotézis már konzisztens, azaz egyetlen negatív példát sem fed már le.

A SPECTRE hasonlóan a FOIL tanuló algoritmushoz, egy kiindulási klózt specializál. Fontos különbség, hogy ezt a FOIL új literál hozzáadásával valósítja meg, míg a SPECTRE egyes klózok törlésével. A SPECTRE algoritmusban minden példa predikátuma azonos az ún. cél predikátum. A kiindulási hipotézisben kell legyen a *cél predikátum definíciója* (2.4. definíció i.) pontja). Egy a cél predikátum definíciójához tartozó *klóz* törölhető, ha csak negatív példákat fed le. A klóz törlés a sikeres levezetések (sikertelen cáfolatok) SLD-fájának [Lloy93] egyszerűsítését, nyesését jelenti. Ezt úgy kell elvégezni, hogy a negatív példákat tartalmazó ágakat le kell vágni, a pozitív példákhoz vezető ágakat pedig meg kell őrizni.

Az *unfolding* transzformáció, amely egy klóz egy literáljának szimbolikus helyettesítése (lásd 3.2. definíció) [Tam84] a kiindulási hipotézissel ekvivalens hipotézist állít elő, azonban a *rezolvens* klózok számát növeli. Egyre nagyobb lehet tehát annak az esélye, hogy egy törölhető klózt kapunk. Determinisztikus, nemrekurzív hipotézisek esetén ez garantálható, hiszen az egymás utáni unfolding transzformációk legvégső esetben elvezetnek a *cél predikátum* csupán a pozitív példák halmazából álló extenzionális definíciójához.

Az alábbiakban egy példán keresztül kerül bemutatásra az algoritmus [Bos94]:

```
E+ = { nyer(pikk, 7), nyer(treff, 3) }
E- = { nyer(kör, 5),  nyer(treff, j) }
```

A kiindulási hipotézis pedig:

```
nyer(S, R) :- szín(S), lap(R).
szín(S) :- fekete(S).
szín(S) :- piros(S).
lap(R)  :- szám(R).
lap(R)  :- figura(R).
fekete(pikk).
fekete(treff).
piros(kör).
piros(káró).
szám(2).
...
szám(9).
figura(j).
figura(d).
figura(k).
figura(a).
```

2.8. Ábra. A SPECTRE algoritmus egy kiindulási feladata

A fenti általános hipotézis egyelőre mind a két negatív példát lefedi (valójában bármilyen legális kártyalap adataira sikeres lesz a `reward/2` predikátum lefuttatása). Az első unfolding transzformáció után a

```
nyer(S, R) :- szín(S),   lap(R).
```

[15] A SPECTRE név a SPECialization by Transformation and Elimination rövidítése, a program letölthető a http://www.dsv.su.se/ML/SPECTRE.html Internet címről.

klóz helyett a

```
nyer(S, R) :- fekete(S), lap(R).
nyer(S, R) :- piros(S), lap(R).
```

új klózokat kapjuk. Az első rezolvens fekete kártyalapokat fedi le, a másik klóz pedig a piros lapokat. Látható, hogy a negatív példák között van egy piros kártyalap., ezért a második klóz elhagyásával egy jobb hipotézist kaphatunk. Mivel a második rezolvens csak negatív példákat fed le, ezért minden pozitív példa továbbra is lefedett marad. Töröljük ki a második klózt, majd hajtsunk végre egy újabb unfolding lépést:

```
nyer(S, R) :- fekete(S),   szám(R).
nyer(S, R) :- fekete(S), figura(R).
```

Ebben az esetben ismét a második klóz az, amelyet törölni lehet, mivel csak negatív példákat fed le. Végül eljutottunk egy olyan hipotézishez, amely *teljes* és *konzisztens*.

A SPECTRE algoritmus vázlata a következő:

```
Input:  Egy P Prolog program, és E+ a pozitív, valamint E- a negatív példák halmaza
Output: Egy P' program, amely specializációja a P programnak,
        figyelembe véve az E+ és E--t.
Új_Program = P
while  létezik klóz C ∈ Új_Program ∧ C lefed egy negatív példát  do
begin
  if  C lefed egy pozitív példát
    then
      a C klózon egy unfolding transzformáció végrehajtása
      Új_Program = Új_Program + rezolvensek - {C klóz}
    else
      Új_Program = Új_Program - {C klóz}
end
return  Új_Program
```

2.9. Ábra. A SPECTRE algoritmus vázlata

Az unfolding transzformáció során minden esetben ki kell jelölni egy literált, amelyet helyettesítünk. Ennek a kiválasztása jelentősen befolyásolja az algoritmus további működését. Boström több különböző módszert próbált ki, kezdve az egyszerű balról az első (Prolog) stratégiától a véletlenszerű választásig. A legsikeresebb az a stratégia volt, amikor az ID3-ból vett *maradék bonyolultság* (residual impurity measure) irányította a választást. Az algoritmus minden esetben a cél predikátum klózait transzformálja. Ezzel szemben kételyeket lehet megfogalmazni. A bonyolultabb gyakorlati példákban gyakran nem a cél predikátum definíciója a hibás, hanem a levezetési fában lejjebb található klóz. A hibáért valóban felelős klóz megtalálása és transzformálása nagy mértékben meggyorsíthatná az algoritmus működését. Erre a módosításra azonban a SPECTRE implementációja nem nyújtott lehetőséget. Jelen dolgozat szerzője ezért, egy teljesen új alapokon nyugvó algoritmust dolgozott ki felhasználva egyes ötleteket a SPECTRE módszerből, amelyben ezt a problémát is megoldotta. Ez a tanuló rendszer az IMPUT [AleI97].

A SPECTRE algoritmus nem viseli el a zajt az adatok között, valamint érzékeny arra, hogy a megtalált klózban melyik literált választjuk ki, hogy szimbolikusan helyettesítsük. Azt is meg kell állapítani, hogy a szimbolikus helyettesítés jelentős mértékben csökkenti a keresési tér méretét. Ha teljesen véletlenszerűen választunk literálokat egy klóz jobb oldalára, és szisztematikusan kapcsoljuk egymáshoz a bennük szereplő változókat, akkor belátható, hogy az így kapott keresési tér jóval nagyobb, mint amikor a jobb oldalon lévő néhány literált szimbolikusan helyettesítjük a definíciójával. Vannak olyan algoritmusok, amelyekben a túl sok literál generálása a jobb oldalra korlátozott, emiatt lehetnek olyan jobb oldalak, amelyeket az algoritmus nem tud meghatározni. A SPECTRE esetében ez nem fordulhat elő. Másfelől viszont az is igaz, hogy a SPECTRE módszerből hiányzik az a lehetőség, hogy tetszőleges literált adjon a jobb oldalhoz. Mivel sok esetben a megoldást ez jelenti, ezért ez a módszer egyik fogyatékossága.

Az első SPECTRE verziót továbbiak követték és Boström 1995-ben elkészítette a SPECTRE II-t [Bos95]. Ez a módszer már lehetőséget biztosított rekurzív predikátumok specializációjára is. A későbbiekben elkészült egy további SPECTRE III verzió is [Bos99].

3. Az IMPUT algoritmus

Ebben a fejezetben a szerző által tervezett és implementált IMPUT tanuló algoritmus kerül ismertetésre. A fejezet anyaga megjelent az Elsevier kiadó által megjelentetett Intelligent Data Analysis Journal-ben 1997-ben [AleI97]. Az IMPUT algoritmus más fórumokon is bemutatásra került: a 1995-ben Leuvenben megrendezett ILP95 konferencián [Ale95], majd 1996-ban az Európai Mesterséges Intelligencia Konferencián [Ale96]. Később az IMPUT algoritmus több alkalmazásáról jelent meg publikáció [KókI96], [KókP96], [Kók96], [Kók97], [Ale97], [Hor99].

Az előző fejezetben említett svéd SPECTRE tanuló algoritmus úgy specializálja a kiindulási hipotézist a pozitív és a negatív példáknak megfelelően, hogy két kitüntetett program klóz transzformációt alkalmaz az ún. *unfolding* transzformációt és a *klóz törlést*. Az itt bemutatásra kerülő IMPUT módszer ennek az algoritmusnak egy módosított változata, amely az IDTS algoritmikus hibakereső (algorithmic debugging) algoritmus és a SPECTRE kombinációjának tekinthető. Az IMPUT rendszer legfontosabb ötlete a következő: a kiindulási Prolog programból ki kell választani azt a program klózt, amelyre alkalmazzuk az unfolding transzformációt. Míg a SPECTRE [Bos94] az első, a levezetésben használt program klózt választja ki, addig az IMPUT az IDTS algoritmikus hibakereső algoritmussal megkeresi a *rossz*, azaz a hibáért valóban felelős program klózt és arra hajtja végre a transzformációt. A transzformálandó klóz kiválasztása kulcsfontosságú az eredmény szempontjából. Bár a SPECTRE algoritmus is mindig meg tud határozni egy a kiindulási példákkal konzisztens megoldást, az eredményül kapott hipotézis méretében, kompaktságában jelentős volt a különbség. Az IMPUT módszer segítségével – a hibakereséskor a rendszerbe pótlólagosan bevitt információknak köszönhetően – a kapott hipotézis mérete jóval kisebbnek adódott, mint a SPECTRE módszer esetében.

A logikai programok specializációja a pozitív és negatív példák alapján tekinthető a programhoz tartozó SLD-fa [Lloy93] egyszerűsítésének (nyesésének) úgy, hogy a negatív példák levezetéseit mind kivágjuk, ellenben minden pozitív példa levezetését megtartjuk. A gyakorlatban a vágás az unfolding transzformációval vagy egy klóz törlésével valósítható meg. Ez így történt a SPECTRE algoritmusban is [Bos94]. A kiindulási hipotézisben lévő cél predikátumot specializálta úgy, hogy három különböző stratégia szerint (balról a legelső, véletlenszerű, bonyolultsági mérték) kiválasztott literálon hajtotta végre az unfolding transzformációt.

A bemutatásra kerülő IMPUT algoritmus alkalmas egy kezdeti hipotézis több predikátumának *interaktív* revíziójára. Alapvetően két részből áll: az IDTS algoritmikus hibakereső módszerből [Hor93], [Paa94] és a SPECTRE algoritmusból. Az alapötlete az, hogy a SPECTRE algoritmust kibővítve egy interaktív hibakereső módszerrel a specializáció minőségében javulás érhető el.

Az IDTS módszer Shapiro eredeti algoritmikus hibakereső módszerének [Shap83] egy módosítása, amely futása során kevesebb kérdést kell, hogy feltegyen a *bölcsnek*. Ha egy eljáráshívás során kapott eredmény helyessége kikövetkeztethető egy CPM (Category-Partition testing Method) konfigurációból, akkor azt a kérdést a rendszer nem teszi fel. További előnye az IDTS módszernek, hogy csak az eredmény szempontjából releváns futási útvonalat vizsgálja [Paa94].

Az IMPUT fő törekvése, hogy az unfolding transzformációnak alávetni kívánt klóz megkeresése kulcs fontosságú a specializáció folyamatának hatékonysága szempontjából. Amennyiben az aktuális hipotézis lefed egy negatív példát, akkor feltevés szerint létezik legalább egy olyan klóz, amely felelős ezért. Amikor az IDTS azonosítja a klózt, az törlésre kerül. Amennyiben egy pozitív példa levezetéséhez az éppen törölt klóz mégis szükséges, akkor egy unfolding transzformációt hajt rajta végre, majd a transzformációval kapott *rezolvens* klózokat a kezdeti hipotézishez hozzáadja. A transzformációnál használt literál kiválasztásához egy módosított bonyolultsági mértéket [Bos94] használ.

A Prolog programok egy speciális osztályát alkotják a formális nyelvekhez készített *elemzők*. A leggyakrabban használt eszköz az ilyen programok írásához a DCG (Definite Clause Grammar) formalizmus [Per80]. A DCG megengedi nemcsak a formális nyelvi átírási szabályok, hanem változók (attribútumok) használatát is a Prolog nyelvben. Az IMPUT rendszer képes DCG szabályok specializálására is, így képes lehet természetes vagy formális nyelvek szintaxisának tanulására, mint például az EKG hullámok vagy más hasonló biológiai hullámok nyelvtana [Kók97].

Az IMPUT rendszer működése különböző bonyolultságú példák segítségével kerül bemutatásra kezdve egy egyszerűvel majd a végén sor kerül a PECG alkalmazásra, amely egy EKG hullám osztályozó rendszer, amelyet egy tanuló modul egészít ki. A 3.1. Fejezetben néhány fontosabb definíció következik, majd magának az algoritmusnak a bemutatása a 3.2. Fejezetben. A 3.3. Fejezetben az IMPUT rendszert egy természetes nyelvi szintaxis (DCG) tanulási feladat megoldására használjuk. A 3.4. Fejezetben a SPECTRE és az IMPUT segítségével kapott teszt eredmények összehasonlítása következik. A 3.5. Fejezetben az EKG hullámok osztályozására készített PECG alkalmazás kerül bemutatásra, végül a 3.6. Fejezetben a hasonló rendszerekkel kapcsolatos összehasonlítás következik.

3.1. Fogalmak, definíciók és elméleti háttér

A következőkben röviden ismertetésre kerülnek a fejezetben használt fogalmak, valamint az IMPUT algoritmus elkészítéséhez felhasznált komponensek.

3.1.1. Definíciók

Legyen P program klózok egy halmaza. Amennyiben a P program lefed egy e negatív példát és a levezetésében szerepel a $p_i \in P$ klóz[16], azonban ez p_i nem szerepel egyetlen pozitív példa levezetésében sem, akkor a p_i klóz törölhető a P programból. Ha a kezdeti hipotézisben eredetileg ilyen elhagyható klózok már nincsenek, akkor újabbak kaphatók az unfolding transzformáció segítségével.

3.1. Definíció: Levezetés [Lloy93]: legyen G_i egy cél (goal) klóz, G_i: $\leftarrow A_1, \ldots A_m, \ldots A_k$ valamint C_{i+1} egy program klóz C_{i+1}: $A \leftarrow B_1, \ldots, B_q$ és R egy levezetési stratégia. Azt mondjuk, hogy G_{i+1} levezethető a G_i-ből és C_{i+1}-ből felhasználva a θ_{i+1} legáltalánosabb egyesítés operátort az R levezetési stratégiával, ha fennállnak a következők:

- Az R által meghatározott literál a G_i-ben A_m
- $A_m\theta_{i+1} = A\theta_{i+1}$
- G_{i+1} a következő cél klóz G_{i+1}: $\leftarrow A_1, \ldots A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots A_k$.

Továbbá a G_{i+1} klóz a G_i és a C_{i+1} *rezolvense.*

3.2. Definíció: Unfolding [Tam84]: legyen p_i program klóz a kiindulási P hipotézisben, p_i: $H \leftarrow A_1, \ldots A_m, \ldots A_k$ és $C = \{c_1, \ldots, c_q\}$ program klózok egy halmaza, $C \subseteq P$ úgy, hogy minden $c_j \in C$ klóz feje azonosítható A_m-mel egy valamilyen θj legáltalánosabb egyesítés operátorral. Az A_m egy R levezetési stratégia által kiválasztott atom. Ekkor az *unfolding* transzformáció a következő:

$$P' = U(P) = P - \{p_i\} \bigcup_{c_j \in C} (H \leftarrow A_1, \ldots, A_{m-1}, body(c_j), A_{m+1}, \ldots A_k$$

Szóban megfogalmazva: a P kezdeti hipotézisből eltávolítjuk a p_i klózt, majd hozzáadjuk a *rezolvenseket.* Valójában az unfolding transzformáció két klóz szimbolikus egymásba-helyettesítését jelenti. Általában az unfolding egyrészt növeli a program klózok számát, másfelől egyre speciálisabb klózokat állít elő (egyre hosszabbak lesznek a klóz törzsek).

A fent említett U unfolding operátort a kezdeti hipotézisre alkalmazva egyre újabb és újabb hipotéziseket kapunk és előbb-utóbb olyan program klózokhoz juthatunk el, amelyek törölhetők anélkül, hogy elrontanák a hipotézis viselkedését a pozitív példákon.

---

[16] A logikai programozásban a levezetés helyett a *refutáció* (cáfolat) elnevezés használatos. A Prolog interpreter a következőképpen működik: feltételezi: a kérdésként kapott cél klóz (goal) nem igaz. A következő lépésben szisztematikusan bejárja a kiindulási klózból dedukcióval kapott ekvivalens formulákat azért, hogy sikerüljön legalább egy ellenpéldát találnia. Ha a cáfolat nem jár sikerrel, akkor az interpreter válasza *igen* (a cél klóz „sajnos" levezethető), ha viszont sikerrel, akkor az interpreter válasza *nem* (azt jelenti, hogy a cél klóz a programból nem vezethető le). A levezethető fogalom itt azt jelenti, hogy a cáfolat nem sikerült, az interpreter *igen*-t válaszolt.

3.3. Definíció: Hibás eljárás (false procedure) [Shap83]: legyen P program klózok egy halmaza és e^- egy tényállítás (negatív példa), M a megtanulandó program modellje[17], amit egy *bölcs* reprezentál. (Az M modell természetesen különbözhet a P aktuális hipotézis által megvalósított M_P legkisebb Herbrand-modelltől.) Feltéve, hogy a P program lefedi e^--t, a p_i klózt *hibás eljárásnak* nevezzük, ha:

- A p_i egy példánya előfordul az e^- cél klóz levezetésében
- A p_i: $B \leftarrow A_1, ..., A_k$ klóz példány törzsében szereplő minden A_i atom benne van az M modellben, de a B nincs.

A modell a P logikai programmal kapcsolatos *igaz* állítások halmaza. A Prolog standard kiszámítási stratégiája (R), amely mindig a balról első literált választja[18] generál egy standard modellt, amit a legkisebb Herbrand-modellnek neveznek és M_P-vel jelölnek. Ettől függetlenül létezik a program M elvárt modellje, amely például a (szóbeli vagy leírt) működési specifikációja. Az elvárt modell nem változik, az M_P viszont változhat annak megfelelően ahogy a P program változik. A tanulás azt jelenti, hogy az elvárt modell alapján egy olyan programot készítünk, amelynek a standard modellje a legjobban megközelíti az elvárt modellt.

3.1.2. A Kategória partícionálás tesztelési eljárás

Az eredeti CPM tesztelési módszert az imperatív programozási nyelvek számára dolgozták ki [Ost88]. A Prolog logikai programozási nyelvre történő átfogalmazása és formális leírása a [Hor93], [Paa94] publikációkban található.

A funkcionális tesztelés során a programokat (eljárásokat) nem lehetséges az input paraméterek minden értékére megvizsgálni. Ezért a tesztelő első feladata a paraméterek kritikus tulajdonságainak definiálása. Ezeket nevezik *kategóriáknak*. A kategóriák osztályokra vannak felbontva (choices). A tesztelés során feltételezik, hogy a végeredménye szempontjából az egy osztályban lévő elemek azonosan viselkednek. Ha egy osztályból kiválasztanak egy teszt esetet, majd erre lefuttatják a programot, akkor siker esetén feltételezik hogy az egész osztályon a program helyesen működik.

Cathegory Measure of Slope

Choice small {abs(slope) < 0,01}
Choice Medium {0,01 abs(slope) 100,0}
Choice Big {100,0 < abs(slope)}
Choice Zero {slope = 0,0} **property** zéró
Choice Inf {slope = inf} **property** végtelen

Cathegory Sign of Slope

Choice negative **if** not zéró and not végtelen
Choice positive

3.1. Ábra. CPM teszt specifikáció a line példához.

A fenti specifikációból a következő teszt keretek állíthatók elő:

{(small, negative), (small, positive), (medium, negative), (medium, positive),
(big, negative),(big, positive), (zero, positive), (inf, positive)}

---

[17] A modellt legegyszerűbb az „igaz" állítások halmazaként elképzelni. Ha rendelkezésre áll egy Prolog program, akkor a programhoz tartozó modell, azon cél klózok halmazát jelenti, amelyek a programmal nem megcáfolhatók (azaz a program logikai következményei). Ha azonban ismert egy modell, az nem jelenti azt, hogy a hozzá tartozó programot is meg tudjuk adni. Lehetnek olyan modellek, amelyekhez nem készíthető program.

[18] A levezetési stratégiák gyakorlati szempontból azonosak [Lloy93]. Azaz, nincs olyan stratégia, amely felhasználásával minden cél klózról véges lépésben eldönthető, hogy az *igaz* vagy *hamis*. Ez azt jelenti, hogy amennyiben a Prolog interpreter sokáig számol, az jelentheti egyrészt a végtelen ciklust is, de azt is hogy még nem ért a levezetési folyamat végére. A levezetési folyamat tetszőleges hosszú lehet. Llyod bebizonyítja, hogy bármely R stratégiát is választunk, mindig lesznek véges idő alatt nem levezethető cél klózok (bár azok stratégiánként lehetnek különbözők).

Ha minden kategóriát definiáltak, akkor az összes teszt keretet (test frame) előállítható. Egy teszt keret minden kategóriából egy osztályt tartalmaz. Gyakran előfordul, hogy a teszt keretek száma végül nagyon nagy lesz. A szükségtelen, vagy értelmetlen teszt keretek el lehet kerülni azzal, ha feltételeket csatolunk az egyes osztályokhoz. Egy osztályt akkor választhatunk be a teszt keretbe, ha a hozzácsatolt *szelektor kifejezés* (selector expression) értéke igaz. A szelektor kifejezésekben tulajdonság azonosítók (property names) vannak. Az osztályokhoz ugyancsak tulajdonság nevek vannak rendelve, amelyeket logikai változóknak lehet tekinteni. A logikai változó értéke igaz, ha az éppen kiválasztott teszt keret tartalmazza azt az osztály, amelyben ez a tulajdonság azonosító jelen van.

A fenti 3.1. ábrán látható példa egy CPM teszt specifikációt mutat be a `line/1` Prolog predikátumra. A példában a `line(m)` egy szakaszt jelöl, amelynek a meredeksége `m`. A `line(inf)` a függőleges szakaszt jelöli, amelynek a meredeksége végtelen. Két kategória van, a meredekség értéke és a meredekség előjele. Mindegyik kategóriában van néhány osztály, valamint van két tulajdonság azonosító: a zéró és a végtelen, ezek segítségével csökkenthető a generált teszt keretek száma.

3.1.3. Az IDTS rendszer

Az IDTS első verziójában [Hor93], [Paa94], [Kók94] Shapiro *hibás eljárás* algoritmusát [Shap83] és a CPM teszt módszert kombinálták [Ost88]. Az IMPUT-ba beépített IDTS-ben már egy fejlettebb algoritmust használnak az ún. *oszd meg és kérdezz* (divide and query) algoritmust [Shap83], amely ugyancsak Shapiro nevéhez fűződik. Az IDTS továbbá programszeletelési (program slicing) módszerekkel is ki van bővítve, amely még bonyolultabbá teszi a működését.

```
fp((A,B),X) :-  fp(A, Y), (Y == ok -> fp(B,X); X = Y).
fp(A,X)     :-  issystem(A) -> (A), X = ok ;
                clause(A, B), fp(B, Y),
                (Y \== ok -> X = Y ;
                query(forall, A, true) -> X = ok; X = (A:-B)).
```

3.2. Ábra. Shapiro *hibás eljárás* algoritmusának [Shap83] Prolog váza

A *hibás eljárás* algoritmus egy lefedett tény (negatív példa) levezetési fáját járja be. A fa csúcspontjaiban kérdéseket tesz fel. A fát post-order irányban (előbb minden leszármazott, azután a szülő csúcs következik) járja be. Amennyiben olyan csúcsot talál, amelyre az összes leszármazottat a *bölcs* helyesnek ítéli, azonban a szülő csúcsot helytelennek, akkor az lesz a hibás klóz. Ha ilyen nincs a fa belsejében, akkor a gyökér lesz a megoldás, mert az éppen egy olyan csúcs, amelyre a leszármazottak helyesek, a szülő (gyökér) pedig nem. A fa bejárása balról jobbra történik és lentről felfelé, ezért a levezetési fában legmélyebben található hibás klózt találja meg az eljárás. Az eljárás a Prolog metainterpreter egy alkalmazása, látszólagos egyszerűsége ellenére minden módosítás komoly háttér ismereteket igényel.

A Shapiro által publikált algoritmikus hibakeresési eljárások lehetővé teszik a hibás eljárás megtalálását, amennyiben adott a program, valamint az a cél klóz, amelyen a program helytelen eredményt ad. Shapiro hibakereső rendszere Prolog programokra készült és a következő típusú hibák megkeresésére szolgált:

- a program terminál, de rossz eredményt számít ki
- a program terminál, de nem számít ki eredményt
- a program nem terminál

A legfőbb hátránya a módszernek az, hogy jelentős számú kérdést tesz fel a *bölcsnek* a közbenső eredmények helyességére vonatkozóan.

Az IDTS hibakereső rendszerben megvalósított módszer, amelyben a CPM tesztelési módszert kombinálták Shapiro algoritmusaival, nagy fejlődést jelentett. A fő elképzelés a következő volt. A *bölcsnek* sok (néha még ráadásul nehéz) kérdésre kell felelni a hibakeresés során. A hibakeresés és a tesztelés általában kapcsolódó tevékenységek. Ha a program már volt tesztelve, akkor annak eredményét a *bölcs* megkérdezése nélkül közvetlenül lehet alkalmazni, mivel a teszt esetek kiértékelését ugyanez a *bölcs* végezte.

Az IDTS rendszer CPM részét akkor lehet használni, ha egy kezdeti teszt adatbázis rendelkezésre áll. A kezdeti teszt specifikáció elkészítése emberi munkával jár, úgy tekinthető mintha további külső tudást vinnénk be a tanuló rendszerbe, amire a hibakereső modulnak van szüksége (és így az inkrementális tanuló algoritmusnak is). Ez nem jelent nagy hátrányt, mivel a tanulás során rendszerint vannak előzetes feltevéseink a megtanulandó

predikátumok viselkedéséről. Ha így tekintjük, akkor a teszt specifikáció a megtanulandó program egy magas szintű leírása, amely nagyon hasonlít az ILP-ben már ismert *integritási feltételekre* (integrity constraint) [DeRa92].

Az IDTS programszeletelési része a Boye által publikált [Boye93] ún. annotáció következtetésen (annotation inference) alapul.[19] Ezt a módszert használva a logikai program funkcionális részéhez automatikusan generálható egy *irányítás* (input vagy output) specifikáció. (Mely változókban vannak az input értékek és melyikben lesz a végrehajtás után az output.) A továbbiakban a felhasználó a program néhány helyét (az eljárások fejében egyes argumentum pozíciókat) a későbbi használat igényeinek megfelelően annotációkkal látja el, végül az IDTS ellenőrzi a teljes funkcionális annotáció helyességét.

Az annotációból egy függőségi gráf (dependency graph) állítható elő a program szeleteléshez. Egy negatív példához tartozó levezetési fát redukálni lehet a függőségi gráf alapján, eltávolítva azokat a részeket, amelyeknek láthatóan nincs befolyásuk a hibára. Az algoritmikus hibakereső algoritmus ez után már csak a fának a *gyanús* részét járja be. A program annotációkat a kezdeti teszt adatbázis elkészítéséhez is fel lehet használni, így a *bölcs* megadhat *teszt eseteket* az annotált program *input* argumentumain keresztül.

A CPM tesztelési módszerrel és program szeletelési technikákkal kibővített IDTS rendszer a Szegedi Tudományegyetem által elvégzett kutatási-fejlesztési feladat volt az „ILP" ESPRIT BRA-6020 projektben. Ez az általános, Prolog programokhoz használható hibakereső és tesztelő eszköz jelentős befolyással bírhat tanuló algoritmusok készítése során [Kók94], amikor egy algoritmus által gyártott hipotéziseket kell hatékonyan tesztelni, illetve kiértékelni. Az IDTS módszer önmagában is alkalmazható volt a PECG [KókP96] alkalmazásban, ahol egy Prologban megvalósított EKG hullám felismerő program hibáit tudta meghatározni. A későbbiekben pedig az IMPUT tanuló algoritmus egyik meghatározó komponense lett. A rendszer kifejlesztésében a szerző is részt vett.

```
rectangle(X,Y,Z,U) :- leftside(X), base(Y), rightside(Z), top(U).

base(X)       :- segment(X).
top(X)        :- segment(X).
leftside(X)   :- segment(X).
rightside(X)  :- segment(X).

segment(X)      :- horiz(X).
segment(X)      :- vert(X).
horiz(line(X)) :- Z is abs(X), Z < 0.01.
vert(line(X))  :- Z is abs(X), Z > 100.

E+ = {
   rectangle(line(150),line(0.005),line(inf),line(0.0)),
   rectangle(line(-160),line(0.0),line(inf),line(0.005)),
   rectangle(line(150),line(-0.004),line(150),line(-0.004)),
   rectangle(line(inf),line(0.005),line(150),line(0.0))
},

E- = {
   rectangle(line(150),line(-160),line(0.005),line(-160)),
   rectangle(line(-170),line(160),line(0.0),line(180)),
   rectangle(line(200),line(0.001),line(300),line(400)),
   rectangle(line(-160),line(0.005),line(-160),line(-160)),
   rectangle(line(0.005),line(0.0),line(0.0),line(0.005)),
   rectangle(line(0.005),line(-160),line(0.0),line(0.0)),
   rectangle(line(150),line(0.0),line(0.0),line(-0.004))
}
```

3.3. Ábra. A `rectangle/4` tanulási feladat az IMPUT algoritmus számára

[19] A módszert eredetileg funkcionális logikai programokhoz fejlesztették ki. Az IDTS-ben (amelyet a Prolog nyelvhez használnak) a funkcionális komponenseket a szokásos logikán kívüli primitívekkel jelölik pl. aritmetikai kifejezések.

3.1.4. A SPECTRE algoritmus

A SPECTRE algoritmus [Bos94] specializálja a logikai programokat a pozitív és a negatív példáknak megfelelően úgy hogy a klóz törlést vagy az unfolding transzformációt alkalmazza [Tam84]. A következőképpen végzi ezt: mindaddig, amíg létezik olyan klóz a programban, amely lefed egy negatív példát, az algoritmus megvizsgálja, hogy ez a klóz lefed-e pozitív példákat. Ha nem fed le egyet sem, akkor törli, egyébként az unfolding transzformációt alkalmazza rá. Az unfolding transzformációban használt literált egy rögzített stratégia alapján választja ki, mivel ennek kulcs fontossága van az algoritmus teljesítménye szempontjából. A kísérleti eredmények azt mutatták [Bos94], hogy a literál kiválasztási stratégiát úgy kell megválasztani, hogy pozitív és a negatív példákat szétválasztó klózok száma minimális legyen. Ez azt jelenti, hogy az unfolding transzformáció alkalmazásainak száma is a lehető legkisebb legyen, mivel a klózok száma az unfolding során általában nő. A literál optimális kiválasztása úgy történik, hogy az a minimális maradék bonyolultságot eredményezze a rezolvensekben az unfolding után. A használt bonyolultsági formula megegyezik az ID3-ban használttal [Quin86].

3.2. Az IMPUT algoritmus ismertetése

Az IMPUT egy abduktív tanuló algoritmus, amely az unfolding transzformációt és a klóz törlést alkalmazza az input hipotézisre miközben figyelembe veszi a pozitív és a negatív példákat. Minden transzformációs lépés után egy *jobb* hipotézist kapunk olyan értelemben, hogy az új hipotézis továbbra is lefedi az összes pozitív példát, a negatív példák közül pedig egyre többet nem fed le. Amikor az algoritmus véget ér az eredményül kapott hipotézis egyetlen negatív példát sem fog lefedni. Sajnálatos módon az algoritmus sikeres befejeződése nem garantált. Cheng és Wolf [Chen96] megvizsgálták az algoritmus elméleti hátterét és javasoltak *teljes* specializációs algoritmust, amely minden esetben megáll és bizonyos minimális feltételek teljesülése esetén esetén megoldást is szolgáltat.

Az IMPUT algoritmus három lépésből áll:

- Egy klóz kiválasztása, amelyre az unfolding transzformációt fogja alkalmazni az algoritmus. Ezt a feladatot az IDTS hibakereső alrendszer végzi.
- A kiválasztott klózon belül egy literál kiválasztása, amelyet az unfolding helyettesíteni fog. Ezt a lépést a SPECTRE algoritmus szerint végzi az IMPUT, a literált az ott megismert bonyolultsági mérték alapján választja ki.
- Végrehajt egy unfolding transzformációt.

```
rectangle(X,Y,Z,U) :- leftside(X), base(Y), rightside(Z), top(U).

base(X)      :- horiz(X).
top(X)       :- horiz(X).
leftside(X)  :- vert(X).
rightside(X) :- vert(X).

segment(X) :-  horiz(X).
segment(X) :-  vert(X).
horiz(line(X)) :- Z is abs(X), Z < 0.01.
vert(line(X))  :- Z is abs(X), Z > 100.
```

3.4. Ábra. A `rectangle/4` tanulási feladat egy elképzelt megoldása

Az IMPUT algoritmus működése a `rectangle/4` példán keresztül kerül bemutatásra. A tanulási feladat a 3.3. Ábrán látható. A program feladat az lenne, hogy felismerjen egy fekvő téglalapot, amelynek bal és jobb oldala függőleges, az alsó és felső oldala vízszintes. Az ábrán látható a kezdeti hipotézis, a pozitív és a negatív példák halmaza. A kezdeti hipotézis felismer illegális téglalapokat is, mint a `rectangle(line(0.005), line(0.0), line(0.0), line(0.005))`, amelynek például négy vízszintes oldala van (a negatív példák halmazában az ötödik). A tanulási folyamat eredménye egy olyan hipotézis lenne, amely minden pozitív példát lefed, és egyetlen negatív példát sem fed le. A következőkben azt láthatjuk, hogy az egymás utáni unfolding transzformációval és klóz törléssel hogyan oldja meg a feladatot az IMPUT algoritmus. A folyamattól azt várjuk, hogy a 3.4. ábrán láthatóhoz hasonló eredményt szolgáltasson.

Az összehasonlítás kedvéért a SPECTRE algoritmus lefuttatásra került a 3.3. ábrán látható feladatra. Az eredmény a 3.5. ábrán látható. Az eredmény világosan bemutatja, hogy a SPECTRE egyik hátránya, hogy csak a

cél predikátumot definiáló egyik klózt tudja transzformálni holott sok esetben (nagyobb programok esetén szinte mindig) erre megfelelőbb lenne egy másik klóz. Természetesen megfelelő számú példával a SPECTRE outputja is tetszőleges pontos lehet. Bár a 3.4. ábrán látható kívánatos megoldást sosem éri el, de azzal ekvivalens eredményt tud adni.

```
rectangle(line(150), line(0.005), A, B)    :- segment(A), horiz(B).
rectangle(line(150), line(-0.004), A, B)   :- rightside(A), top(B).
rectangle(line(-160), line(0.0), A, B)     :- rightside(A), top(B).
rectangle(line(-160), line(-0.004), A, B)  :- rightside(A), top(B).
rectangle(line(inf), A, B, C)              :- base(A), rightside(B), top(C).

base(A)      :- segment(A).
top(A)       :- segment(A).
leftside(A)  :- segment(A).
rightside(A) :- segment(A).

segment(A) :- horiz(A).
segment(A) :- vert(A).
horiz(line(X)) :- Z is abs(X), Z < 0.01.
vert(line(X))  :- Z is abs(X), Z > 100.
```

3.5. Ábra. A SPECTRE algoritmus által adott megoldás a `rectangle/4` tanulási feladatra

Az IMPUT rendszer, amely ötvözi a SPECTRE transzformációit és az IDTS interaktív hibakereső rendszert képes meghatározni a 3.4. ábrán látható megoldást a `rectangle/4` feladatra.

Input: P = { $p_1, p_2, ..., p_u$ } kezdeti hipotézis, B = { $b_1, b_2, ..., b_v$ } háttér tudás, (a háttértudáshoz tartozó klózok nem változnak meg a tanulási folyamat során), valamint E+ és E- változómentes tényekből álló halmazok, a pozitív és negatív példák.

Output: Prolog programok sorozata (hipotézisek), $P'^{(0)}$, $P'^{(1)}$,..., $P'^{(n)}$, ahol $P'^{(0)} = P$, és $P'^{(i+1)} =$ ($P'^{(i)}$), 0 i < n, az az *U* unfolding operátor kiegészítve a klóz törléssel.

```
begin IMPUT
  if  P nem fedi le az összes pozitív példát
     then stop „A kezdeti hipotézis nem fedi le az összes pozitív példát"
  if  option_A
     then összes klóz törlése, amely nem szükséges a pozitív példák levezetéséhez
  let i = 0
  while  létezik olyan e ∈ E⁻ negativ példa, amelyet lefed a P'(i) hipotézis do
  begin
    az IDTS rendszerrel megkeresni azt a c ∈ P'(i) klózt, amely felelős a hibáért
    /*  ilyen klóz mindenképpen kell hogy létezzen  */
    if  c klóz nem fordul elő pozitív példák levezetésében
       then a c klóz törlése, P'(i+1) = P'(i) - {c}
    else
    begin
      az unfolding transzformációhoz legmegfelelőbb literál kiválasztása
      if  nincs ilyen literál
         then stop
      az unfolding transzformáció végrehajtása
      let C = {c1, c2, ..., cs}   a rezolvensek halmaza
      if  option_B
        then a C-ből minden olyan klóz törlése, amelyek nem fordulnak elő
             pozitív példák levezetésében
      let P'(i+1) = P'(i) - {c} ∪ C
    end

    let i = i + 1
  end  while
  return  P'(i)
end IMPUT
```

3.6. Ábra. Az IMPUT algoritmus formális leírása

3.2.1. Az algoritmus formális leírása

Az IMPUT algoritmus alapötlete az, hogy a *„hibás"* klóz kiválasztása kulcsfontosságú a specializálási folyamat hatékonysága szempontjából. Azt feltételezi, hogy ha egy közbenső hipotézis lefed egy adott negatív példát, akkor biztosan létezik egy olyan klóz, amely felelős ezért. A *„hibás"* klóz megtalálására az IMPUT az IDTS interaktív hibakereső alrendszert használja. A megtalált klózt fogja majd transzformálni az algoritmus a következő lépésben.

Az IMPUT rendszerben hasonlóan más ILP rendszerekhez megengedett a háttértudás használata, amely a specializáció folyamán változatlan formában megmarad. Ezt úgy valósítja meg, hogy minden input klózhoz tartozik egy címke, amely megmutatja hogy a háttér tudáshoz tartozik-e. A háttér tudáshoz tartozó klózokat az IDTS hibakereső rendszer nem vizsgálja, ezért az IMPUT sem is fogja transzformálni őket. Emiatt a háttértudás változatlan formában megmarad a tanulási folyamat során.

A klóz törlésre nézve van egy választási lehetőség az algoritmusban: vagy minden klózt törlünk, amit egyáltalán lehet és így egy *leginkább specifikus* (most specific) programhoz jutunk, vagy csak azokat a klózokat töröljük, amely eltávolítása feltétlenül szükséges és így egy *legkevésbé specifikus* (least specific) programot kapunk. Mindkét esetben a kapott hipotézis teljes és konzisztens lesz, azaz minden pozitív példát lefed, de nem fed le egyetlen negatív példát sem. Ezek a választási lehetőségek szerepelnek az algoritmusban. Ha sem az option_A, sem az option_B nincs bekapcsolva, akkor a *legkevésbé specifikus* programot kapjuk. A következő fejezetben bemutatjuk az algoritmus működését a `rectangle/4` példán olyan opciókkal, hogy az a *legkevésbé specifikus* programot állítsa elő.

A kiindulási hipotézis:

```
rectangle(X,Y,Z,U) :- leftside(X), base(Y), rightside(Z), top(U).

base(X)      :- segment(X).
top(X)       :- segment(X).
leftside(X)  :- segment(X).
rightside(X) :- segment(X).
segment(X)   :- horiz(X).
segment(X)   :- vert(X).
```

A háttér tudás:

```
horiz(line(X)) :- Z is abs(X), Z < 0.01.
vert(line(X))  :- Z is abs(X), Z > 100.
```

3.7. Ábra. A módosított `rectangle/4` feladat

3.2.2. Az IMPUT algoritmus működése a `rectangle/4` példán

Ebben a fejezetben az IMPUT rendszer működése részletesen is bemutatásra kerül a 3.3. ábrán bemutatott `rectangle/4` példán keresztül. A példa egy teszt adatbázis használatát is bemutatja a hibakeresés folyamán. A `horiz/1` és a `vert/1` predikátumok a háttér tudáshoz tartoznak. Ez a megkülönböztetés a hibakeresés közben feltett kérdések számának csökkenését eredményezi, ugyanis háttér klózokat a hibakereső rendszer eleve helyeseknek tételezi fel. A 3.7. ábrán látható a módosított kiindulási feladat.

A programhoz tartozó CPM teszt specifikáció a 3.1. ábrán látható a 3.1.2. fejezetben. A `segment/1` predikátum egyetlen argumentuma a tartó egyenesének a meredeksége: az 50 meredekségű szakaszt a `segment(line(50))` tény reprezentálja. A teszt specifikáció eredetileg egy `line/1` predikátumra készült, de mivel a `segment/1` argumentuma éppen egy egyenes (`line`), azért felhasználható változtatás nélkül.

A teszt adatbázis egy adott predikátumra és *teszt keretekre* teszt eseteket tartalmaz. Egy teszt eset tartalmaz: input paramétereket, a program lefuttatásának eredményét (ez egy logikai érték; ha a program lefedi a teszt esetet akkor *igaz*, egyébként *hamis*), fedés esetén a visszakapott output változók értékét, továbbá a teszt eset egy kiértékelését, amelyet egy *bölcs* ad meg. A kiértékelés lehet nem definiált, korrekt és inkorrekt. A teszt adatbázis reprezentálható Prolog tények sorozatával. Minden egyes tény megfelel egy predikátumnak és az egyik hozzá tartozó *teszt keretnek*. A teszt esetek adatai: az input paraméterek, az eredmény, a bölcs által adott kiértékelés, valamint az aggregált (az egész teszt keretre vonatkozó összesített) kiértékelés a `test/6` predikátum

argumentumaiban vannak tárolva. A teszt adatbázis feltöltését az IMPUT rendszertől függetlenül végezheti el a felhasználó egy tesztelő program segítségével. Legyen a teszt adatbázis tartalma a következő (az adatokat korábban a tesztelő program futtatásával, és a kapott eredmény kiértékelésével nyerte a felhasználó):

```
test(segment,1,[small,negative],[line(-0.001)],true,correct).
test(segment,1,[small,positive],[line(0.002)],true,correct).
test(segment,1,[medium,negative], _, _, undefined).
test(segment,1,[medium,positive], _, _, undefined).
test(segment,1,[big,negative], _, _, undefined).
test(segment,1,[big,positive], _, _, undefined).
test(segment,1,[zero,positive],[line(0)],true,correct).
test(segment,1,[inf,positive], _, _, undefined).
```

3.8. Ábra. Példa teszt adatbázis az IDTS alrendszerhez

Például a `test(segment, 1, [small,negative], [line(-0.001)], true, correct)` tény a a `segment/1` predikátum (első két argumentum) `[small,negative]` (harmadik argumentum) *teszt keretéhez* tartozó következő információkat tartalmazza: a teszt keret a `line(-0.001)` *teszt esetet* tartalmazza (negyedik argumentum). Ezt az input paramétert megadva a `segment/1` predikátumnak az eredmény *true* (ötödik argumentum) azaz lefedi a program ezt a példát, az aggregált kiértékelés pedig *correct*, ahogyan azt a *bölcs* kiértékelte (hatodik paraméter). A fenti teszt adatbázis olyan teszt esetekre tartalmaz kiértékelést, amelyek meredeksége *0,0* vagy kis pozitív illetve negatív szám. Az esetek mindegyikében az aggregált kiértékelés *correct* volt. A többi teszt kerethez nem tartoznak teszt esetek, és az aggregált kiértékelés is *undefined*. A `segment/1` predikátum tekinthető a `rectangle/4` program egyik segéd eljárásának, amelyhez részleges teszt eredmények állnak rendelkezésre. A `segment/1` predikátum, amely vagy vízszintes vagy függőleges egyeneseket ismer fel, nem tartalmaz hibát. A teszt adatbázis jelen esetben csupán arra jó, hogy tovább csökkentse az IDTS hibakereső alrendszer által feltett kérdések számát. Az IDTS a teszt adatbázisban nyilvántartott teszt kerethez tartozó input értékek esetén nem tesz fel kérdést a bölcsnek, hanem az aggregált kiértékeléssel fut tovább, ha ilyen van. Nem szükséges, hogy az input argumentum pontosan megegyezzen az adatbázisbeli teszt esettel, elegendő ha ugyanahhoz a teszt kerethez tartoznak, ez a CPM tesztelés legfontosabb alapfeltevése. A továbbiakban végrehajtjuk a 3.6. ábrán bemutatott algoritmust.

```
rectangle(X,Y,Z,U) :- leftside(X), base(Y), rightside(Z), top(U).

base(X)      :- horiz(X).
top(X)       :- segment(X).
leftside(X)  :- segment(X).
rightside(X) :- segment(X).
segment(X)   :- horiz(X).
segment(X)   :- vert(X).
```

3.9. Ábra. A $P'^{(1)}$ hipotézis az IMPUT első lépése után

i = 0

A kezdeti hipotézis $P'^{(0)}$ minden pozitív példát lefed. Az első negatív példát `rectangle(line(150), line(-160), line(0.005), line(-160))` az E'-ból is lefedi a $P'^{(0)}$ hipotézis, ezért az IDTS alrendszer indul el, hogy megkeresse a hibáért felelős klózt. A következő kérdéseket teszi fel a *bölcsnek* (a háttér tudásban lévő predikátumok helyességére nem tesz fel kérdést, és azokra a tényekre sem, amelyekhez tartozó teszt kerethez a *korrect* aggregált kiértékelés tartozik):

```
segment(line(150)) is it ok? (y/n) y
leftside(line(150)) is it ok? (y/n) y
segment(line(-160)) is it ok? (y/n) y
base(line(-160)) is it ok? (y/n) n
```

Kövessük most a 3.6. ábrán bemutatott algoritmus lépéseit. Az IDTS hibakereső alrendszer megtalálta a hibás működésért felelős klózt. Ezután a klózt az IMPUT specializálja az unfolding transzformációval. Mivel a `base(X) :- segment(X)` klóz jobb oldalán egyetlen literál található, ezért ezt fogja helyettesíteni. A rezolvensek a következők: C = { `base(X) :- horiz(X)` ; `base(X) :- vert(X)` }. A második rezolvenst (`base(X) :- vert(X)`) a rendszer törli, mert nincs olyan pozitív példa, amelynek a levezetéséhez szükség lenne erre. Az első hipotézis $P'^{(1)}$ a 3.9. ábrán látható.

Mielőtt a második lépésre sor kerülne, vizsgáljuk meg a teszt adatbázist. Megjegyezzük, hogy az IDTS alrendszer a hibakeresés közben tudomására jutott kiértékeléseket folyamatosan tárolja a teszt adatbázisba. Úgy is lehet mondani, hogy hibakeresés közben tesztel is.

```
test(segment,1,[small,negative],[line(-0.001)],true,correct).
test(segment,1,[small,positive],[line(0.002)],true,correct).
test(segment,1,[medium,negative], _, _, undefined).
test(segment,1,[medium,positive], _, _, undefined).
test(segment,1,[big,negative],[line(-160)],true,correct).
test(segment,1,[big,positive],[line(150)],true,correct).
test(segment,1,[zero,positive],[line(0)],true,correct).
test(segment,1,[inf,positive], _, _, undefined).
```

3.10. Ábra. A teszt adatbázis az IMPUT első lépése után

A teszt adatbázis tartalma a 3.10. ábrán látható. Két teszt keret adatai változtak meg: a nagy pozitív és a nagy negatív meredekséghez tartozó adatok. Ezek a potenciális teszt esetek a hibakeresés közben kerültek elő, és – ha már a bölcs megválaszolta őket – a válaszokat az IDTS el is tárolta. A továbbiakban ezekre a teszt keretekre már nem fog kérdéseket feltenni. A még hiányzó keretek a (medium, negative), (medium, positive), (inf, positive). Az IDTS hibakereső alrendszer megjegyzi azokat a sporadikus tényekre adott kiértékeléseket is, amelyeket nem tud tárolni a teszt adatbázisban.

i = 1

A $P'^{(1)}$ hipotézis már nem fedi le az első, a második és a hatodik negatív példát:

```
rectangle(line(150),line(-160),line(0.005),line(-160)).
rectangle(line(-170),line(160),line(0.0),line(180)).
rectangle(line(0.005),line(-160),line(0.0),line(0.0)).
```

viszont lefedi a harmadik negatív példát:

```
rectangle(line(200),line(0.001),line(300),line(400)).
```

A hibakereső alrendszer a következő kérdéseket teszi fel:

```
leftside(line(150)) is it ok? (y/n) y    ezt a választ az IDTS tárolta, ezért nem kérdezi meg ismét
base(line(0.001)) is it ok? (y/n) y
rightside(line(300)) is it ok? (y/n) y
top(line(400)) is it ok? (y/n) n
```

A következő specializálandó hibás klóz a `top(X) :- segment(X)`. A jobb oldalon ismét csak egy literál található, ezért azt fogja a rendszer helyettesíteni. A rezolvensek a következők: C = {`top(X) :- horiz(X)`; `top(X) :- vert(X)`}. A második klóz nem fordul elő pozitív példák levezetésében, ezért az IMPUT eltávolítja C-ből. Az eredmény a 3.11. ábrán látható.

i = 2

A $P'^{(2)}$ hipotézis már öt negatív példát nem fed le: az első négyet és a hatodikat, azonban lefedi az ötödiket:

```
rectangle(line(0.005),line(0.0),line(0.0),line(0.005))
```

```
rectangle(X,Y,Z,U) :- leftside(X), base(Y), rightside(Z), top(U).

base(X)      :- horiz(X).
top(X)       :- horiz(X).
leftside(X)  :- segment(X).
rightside(X) :- segment(X).
segment(X)   :- horiz(X).
segment(X)   :- vert(X).
```

3.11. Ábra. A $P'^{(2)}$ hipotézis az IMPUT második lépése után

Az IDTS által megtalált hibás klóz most a `leftside(X) :- segment(X)`, amelyre az unfolding transzformációt kell alkalmazni. Az előzőekhez hasonlóan kapjuk a $P'^{(3)}$ hipotézist. Végül már csak egyetlen negatív példát fed le a hipotézis, a hetediket.

```
rectangle(line(150),line(0.0), line(0.0),line(-0.004)).
```

Az IDTS rendszer által talált klóz a `rightside(X) :- segment(X)`, amelyre kell az unfolding transzformációt alkalmazni. A $P'^{(4)}$ hipotézis a 3.12. ábrán látható. Ez a program már teljes és konzisztens, az E^+ minden elemét lefedi, és egyetlen elemet sem fed le az E^--ból. **Az itt látható program teljes egészében azonos a 3.4. ábrán bemutatott elképzelt megoldással.**

```
rectangle(X,Y,Z,U) :- leftside(X), base(Y), rightside(Z), top(U).

base(X)      :- horiz(X).
top(X)       :- horiz(X).
leftside(X)  :- vert(X).
rightside(X) :- vert(X).
segment(X)   :- horiz(X).
segment(X)   :- vert(X).
```

3.12. Ábra. A $P'^{(4)}$ hipotézis az IMPUT végeredménye a `rectangle/4` példára

Az IMPUT algoritmus egy kényelmetlensége, hogy működéséhez igényli egy *bölcs* közreműködését, akinek a hibakeresés közben kérdésekre kell válaszolnia, hogy a hibás klózt azonosítani tudja. Az IDTS rendszer a lehetőségekhez mérten azonban az ilyen kérdések számát jelentősen csökkenteni tudja.

3.3. Egyszerű angol mondatok szintaxisának tanulása

Ebben a fejezetben egy további példán keresztül kerül bemutatásra az IMPUT algoritmus. A példa neve: `sentence`. A példában szereplő program egyszerű angol mondatok elemzésére szolgál, lásd a 3.13. ábrát. Mint az látható a kezdeti program DCG (Definite Clause Grammar) [Per80] szabályokkal adott, azaz formális nyelvi átírási szabályokból áll, nem Prolog klózokból.

Az alábbiakban az angol természetes nyelv szintaxisához kötődő egyszerű példa ismertetésére kerül sor. A `sentence` nyelvtan olyan angol mondatokat ír le, amelyekben egy egyszerű igei és egy egyszerű főnévi szerkezet található. Mind a kettő lehet egyes vagy többes számú. A példa kedvéért a kezdeti hipotézis nem egyezteti az igei és a főnévi szerkezetet egymással. A továbbiakban azt mutatjuk be, hogy az IMPUT hogyan tanulja meg a példákból a hipotézis korrekt verzióját, amely egyezteti az egyes számú igét az egyes számú főnévi szerkezettel, a többes számú igét a többes számú főnévi szerkezettel.

A `sentence` példához hét pozitív és három negatív példa tartozik. Az utolsó négy pozitív példa csak technikai jelentőséggel bír, ezen a módon kényszeríteni lehet az IMPUT rendszert, hogy az unfolding transzformációt alkalmazza a `noun_phrase` vagy a `verb_phrase` predikátum klózainak törlése helyett.

```
0: sentence --> nounphrase, verbphrase.
1: nounphrase --> singular_nounphrase.
2: nounphrase --> plural_nounphrase.
3: singular_nounphrase --> nm.
4: singular_nounphrase --> singular_determiner, singular_noun.
5: plural_nounphrase --> plural_determiner, plural_noun.
6: plural_nounphrase --> plural_noun.
7: verbphrase --> singular_verb.
8: verbphrase --> plural_verb.

E+ = {sentence([bob,plays],[]),        sentence([dogs,run],[]),
      sentence([the,boy,jumps],[]),    nounphrase([bob],[]),
      nounphrase([the,dogs],[]),       verbphrase([run],[]),
      verbphrase([runs],[])},

E- = {sentence([the,boys,plays],[]), sentence([boys,runs],[]),
      sentence([a,dog,jump],[])}
```

3.13. Ábra. A kiindulási hipotézis a `sentence` példához

Elindulva az első negatív példából: „*The boys plays*" a beépített IDTS hibakereső rendszer azt találja, hogy a `sentence --> nounphrase, verbphrase` klóz a felelős a hibáért. A bonyolultsági stratégiának megfelelően kiválasztja az első literált, majd végrehajtja az unfolding transzformációt. A rezolvensek *C* halmaza a következő lesz:

```
C = {
     Sentence --> singular_nounphrase, verbphrase.
     Sentence --> plural_nounphrase, verbphrase.
}
```

Ezután az IMPUT törli a hibás klózt, és a helyére beilleszti a rezolvenseket. Az algoritmus ebben a fázisban nem tud egyetlen klózt sem törölni (a pozitív példák lefedettségét a hipotézis megőrzi), így az új hipotézis a `0`-s számú klóz helyett két új klózt fog tartalmazni, a `0-1` és a `0-2`-s számút, a többi klóz változatlan marad.

```
0-1: sentence --> singular_nounphrase, verbphrase.
0-2: sentence --> plural_nounphrase, verbphrase.
  1: nounphrase --> singular_nounphrase.
  2: nounphrase --> plural_nounphrase.
  3: singular_nounphrase --> nm.
  4: singular_nounphrase --> singular_determiner, singular_noun.
  5: plural_nounphrase --> plural_determiner, plural_noun.
  6: plural_nounphrase --> plural_noun.
  7: verbphrase --> singular_verb.
  8: verbphrase --> plural_verb.
```

3.14. Ábra. A `sentence` példa az első lépés végrehajtása után

A jobb érthetőség miatt az IMPUT rendszer a klózokat sorszámmal látja el. Az unfolding transzformációval kapott klózt már egy szám pár fogja azonosítani. A szám pár első eleme a kiindulási klóz sorszáma, míg a második szám a behelyettesített klóz sorszáma. A literál pozíciót, ahol a transzformációt a rendszer végrehajtja nem tárolja. Általában ez magától értetődő. A `0-1` szám egy olyan klózt jelöl, amely a `0` sorszámú klózból keletkezett olymódon, hogy valamelyik literálja helyére behelyettesítette `1` sorszámú klózt.

Amennyiben klóz törlésre nem kerül sor, a kapott hipotézis ekvivalens az eredeti hipotézissel. Ez történik most is. A 3.14. ábrán látható hipotézis továbbra is lefedi az első negatív példát („*The boys plays*"). Ezért a `sentence --> plural_nounphrase, verbphrase` klóz a felelős. Erre a klózra fogja az IMPUT az unfolding transzformációt alkalmazni, azon belül is a második literálra. A rezolvensek *C* halmaza a következő lesz:

```
C = {
     Sentence --> plural_nounphrase, singular_verb.
     Sentence --> plural_nounphrase, plural_verb.
}
```

A rezolvensek közül az első klózt lehet törölni, mivel az csak negatív példát fed le. Miután az algoritmus a hibás klózt helyettesítette a rezolvenseivel, megkapjuk a 3.15. ábrán látható hipotézist.

```
0-1: sentence --> singular_nounphrase, verbphrase.
0-2-8: sentence --> plural_nounphrase, plural_verb.
  1: nounphrase --> singular_nounphrase.
  2: nounphrase --> plural_nounphrase.
  3: singular_nounphrase --> nm.
  4: singular_nounphrase --> singular_determiner, singular_noun.
  5: plural_nounphrase --> plural_determiner, plural_noun.
  6: plural_nounphrase --> plural_noun.
  7: verbphrase --> singular_verb.
  8: verbphrase --> plural_verb.
```

3.15. Ábra. A `sentence` példa az második lépés végrehajtása után

Az itt látható hipotézis már csupán egyetlen negatív példát fed le („A dog jump"). A hibakereső rendszer által megtalált felelős klóz: `sentence --> singular_nounphrase, verbphrase`. Az unfolding transzformáció után a rezolvensek a következők:

```
C = {
     Sentence --> singular_nounphrase, singular_verb.
     Sentence --> singular_nounphrase, plural_verb.
}
```

A rezolvensek közül a második klózt lehet törölni, mivel az csak negatív példát fed le. Miután az algoritmus a hibás klózt helyettesítette a rezolvenseivel, megkapjuk a 3.16. ábrán látható végeredményt.

```
0-1-7: sentence --> singular_nounphrase, singular_verb.
0-2-8: sentence --> plural_nounphrase, plural_verb.
  1: nounphrase --> singular_nounphrase.
  2: nounphrase --> plural_nounphrase.
  3: singular_nounphrase --> nm.
  4: singular_nounphrase --> singular_determiner, singular_noun.
  5: plural_nounphrase --> plural_determiner, plural_noun.
  6: plural_nounphrase --> plural_noun.
  7: verbphrase --> singular_verb.
  8: verbphrase --> plural_verb.
```

3.16. Ábra. A sentence példa végeredménye

A fenti ábrán látható hogy végül az IMPUT tanuló algoritmus a hibás nyelvtant kijavította. Ehhez szüksége volt egy *bölcs* segítségére, aki a hibakeresés során a hibás klóz megtalálásához nyújtott segítséget. A tanuló rendszer interaktív használata lehetővé teszi, hogy a *bölcs* több futtatást is végezzen, és a válaszainak az eredményül kapott hipotézisre gyakorolt hatását megértse. A teljes tanulási folyamat számítógépes outputja az alábbiakban megtekinthető:

```
| ?- start.
Welcome to IMPUT learning system.
Please enter the filename to be processed: sentence.

 The background knowledge is:

 9: nm([bob|A],A)
10: singular_determiner([a|A],A)
11: singular_determiner([the|A],A)
12: plural_determiner([the|A],A)
13: singular_noun([boy|A],A)
14: singular_noun([dog|A],A)
15: plural_noun([boys|A],A)
16: plural_noun([dogs|A],A)
17: singular_verb([runs|A],A)
18: singular_verb([plays|A],A)
19: singular_verb([jumps|A],A)
20: plural_verb([run|A],A)
21: plural_verb([play|A],A)
22: plural_verb([jump|A],A)

 The theory needs to be specialized is:

0: sentence(A,B):-nounphrase(A,C), verbphrase(C,B)
1: nounphrase(A,B):-singular_nounphrase(A,B)
2: nounphrase(A,B):-plural_nounphrase(A,B)
3: singular_nounphrase(A,B):-nm(A,B)
4: singular_nounphrase(A,B):-singular_determiner(A,C), singular_noun(C,B)
5: plural_nounphrase(A,B):-plural_determiner(A,C), plural_noun(C,B)
6: plural_nounphrase(A,B):-plural_noun(A,B)
7: verbphrase(A,B):-singular_verb(A,B)
8: verbphrase(A,B):-plural_verb(A,B)

 The positive examples are:

23: sentence([bob,plays],[])
24: sentence([dogs,run],[])
25: sentence([the,boy,jumps],[])
26: nounphrase([bob],[])
27: nounphrase([the,dogs],[])
28: verbphrase([run],[])
29: verbphrase([runs],[])

 The negative examples are:

30: sentence([the,boys,plays],[])
31: sentence([boys,runs],[])
32: sentence([a,dog,jump],[])
```

```
Checking input examples:

 The sets of positive and negative examples are distinct.

 Checking positive examples:

 Checking negative examples:

30: sentence([the,boys,plays],[]) covered.
31: sentence([boys,runs],[]) covered.
32: sentence([a,dog,jump],[]) covered.

 The fact sentence([the,boys,plays],[]) is covered by the theory.
 Starting the false proc. algorithm to determine the basis of the unfolding.

 Is it ok [plural_nounphrase([the,boys,plays],[plays])] (y/n) y
 Is it ok [nounphrase([the,boys,plays],[plays])] (y/n) y
 Is it ok [verbphrase([plays],[])] (y/n) y
 Is it ok [sentence([the,boys,plays],[])] (y/n) n

 Unfolding at the clause instance:
   0: sentence([the,boys,plays],[]):-
      nounphrase([the,boys,plays],[plays]),
      verbphrase([plays],[])

 - trying resolvent(s): [0-1]
          actual minimum is: 10.18355343004271.
 - trying resolvent(s): [0-2]
           actual minimum is: 10.390359525563188.

 The result of the unfolding is:

0-1: sentence(A,B):-singular_nounphrase(A,C), verbphrase(C,B)
0-2: sentence(A,B):-plural_nounphrase(A,C), verbphrase(C,B)
1: nounphrase(A,B):-singular_nounphrase(A,B)
2: nounphrase(A,B):-plural_nounphrase(A,B)
3: singular_nounphrase(A,B):-nm(A,B)
4: singular_nounphrase(A,B):-singular_determiner(A,C), singular_noun(C,B)
5: plural_nounphrase(A,B):-plural_determiner(A,C), plural_noun(C,B)
6: plural_nounphrase(A,B):-plural_noun(A,B)
7: verbphrase(A,B):-singular_verb(A,B)
8: verbphrase(A,B):-plural_verb(A,B)

 Checking positive examples:

 Checking negative examples:

30: sentence([the,boys,plays],[]) covered.
31: sentence([boys,runs],[]) covered.
32: sentence([a,dog,jump],[]) covered.

 The above theory:
    cover   7 positive samples from 7 (100.00%) and
    fail on 0 negative samples from 3 (0.00%).

 The fact sentence([the,boys,plays],[]) is covered by the theory.
 Starting the false proc. algorithm to determine the basis of the unfolding.

 Unfolding at the clause instance:
   0-2: sentence([the,boys,plays],[]):-
        plural_nounphrase([the,boys,plays], [plays]),
        verbphrase([plays],[])

 - trying resolvent(s): [0-2-1]
          actual minimum is: 13.368065554250645.
 - trying resolvent(s): [0-2-2]
          actual minimum is: 12.613178052087179.
```

```
 The result of the unfolding is:

0-1: sentence(A,B):-singular_nounphrase(A,C), verbphrase(C,B)
0-2-8: sentence(A,B):-plural_nounphrase(A,C), plural_verb(C,B)
1: nounphrase(A,B):-singular_nounphrase(A,B)
2: nounphrase(A,B):-plural_nounphrase(A,B)
3: singular_nounphrase(A,B):-nm(A,B)
4: singular_nounphrase(A,B):-singular_determiner(A,C), singular_noun(C,B)
5: plural_nounphrase(A,B):-plural_determiner(A,C), plural_noun(C,B)
6: plural_nounphrase(A,B):-plural_noun(A,B)
7: verbphrase(A,B):-singular_verb(A,B)
8: verbphrase(A,B):-plural_verb(A,B)

 Checking positive examples:


 Checking negative examples:

32: sentence([a,dog,jump],[]) covered.

 The above theory:
    cover   7 positive samples from 7 (100.00%) and
    fail on 2 negative samples from 3 (66.67%).

 The fact sentence([a,dog,jump],[]) is covered by the theory.
 Starting the false proc. algorithm to determine the basis of the unfolding.

 Is it ok [singular_nounphrase([a,dog,jump],[jump])] (y/n) y
 Is it ok [verbphrase([jump],[])] (y/n) y
 Is it ok [sentence([a,dog,jump],[])] (y/n) n

 Unfolding at the clause instance:
   0-1: sentence([a,dog,jump],[]):-
        singular_nounphrase([a,dog,jump], [jump]),
        verbphrase([jump],[])

 - trying resolvent(s): [0-1-1]
          actual minimum is: 4.141709450076293.
 - trying resolvent(s): [0-1-2]
          actual minimum is: 3.900134529890125.

 The result of the unfolding is:

0-1-7: sentence(A,B):-singular_nounphrase(A,C), singular_verb(C,B)
0-2-8: sentence(A,B):-plural_nounphrase(A,C), plural_verb(C,B)
1: nounphrase(A,B):-singular_nounphrase(A,B)
2: nounphrase(A,B):-plural_nounphrase(A,B)
3: singular_nounphrase(A,B):-nm(A,B)
4: singular_nounphrase(A,B):-singular_determiner(A,C), singular_noun(C,B)
5: plural_nounphrase(A,B):-plural_determiner(A,C), plural_noun(C,B)
6: plural_nounphrase(A,B):-plural_noun(A,B)
7: verbphrase(A,B):-singular_verb(A,B)
8: verbphrase(A,B):-plural_verb(A,B)

 Checking positive examples:


 Checking negative examples:


 The above theory:
    cover   7 positive samples from 7 (100.00%) and
    fail on 3 negative samples from 3 (100.00%).

 ---------------- The final result theory is: ----------------

0-1-7: sentence(A,B):-singular_nounphrase(A,C), singular_verb(C,B)
0-2-8: sentence(A,B):-plural_nounphrase(A,C), plural_verb(C,B)
```

```
1: nounphrase(A,B):-singular_nounphrase(A,B)
2: nounphrase(A,B):-plural_nounphrase(A,B)
3: singular_nounphrase(A,B):-nm(A,B)
4: singular_nounphrase(A,B):-singular_determiner(A,C), singular_noun(C,B)
5: plural_nounphrase(A,B):-plural_determiner(A,C), plural_noun(C,B)
6: plural_nounphrase(A,B):-plural_noun(A,B)
7: verbphrase(A,B):-singular_verb(A,B)
8: verbphrase(A,B):-plural_verb(A,B)
```

3.4. Az IMPUT rendszer alkalmazásával kapott kísérleti eredmények

Az IMPUT rendszer tesztelésére három különböző tanulási feladatot használtunk. A három példa a következő volt: `rectangle`, `sentence`, `shuttle`. Az első kettő megtalálható a 3.2. és a 3.3. Fejezetekben. A `sentence` és a `shuttle` példa egy tanuló programok számára készített nemzetközi adatbázisból származik. A `shuttle` példa ismertetése megtalálható a [Bos94] publikációban. Az eredményül kapott Prolog klózok számát és a kapott program pontosságát vizsgáltuk.

A tesztfeladat során kapott eredmények Az IMPUT tanuló algoritmussal, a Prolog stratégia felhasználásával				
Feladat	A program klózok száma	Pontosság		
		Pozitív példákon	Negatív példákon	Összesítve
Rectangle	5.0	100,00%	84,62%	92,21%
Sentence	5.5	46,34%	85,00%	65,43%
Shuttle	18.6	65,58%	75,58%	70,81%

3.17. Táblázat. Az IMPUT algoritmus eredményei a Prolog stratégia felhasználásával

A tesztfeladat során kapott eredmények A SPECTRE tanuló algoritmussal, a bonyolultsági mérték stratégia felhasználásával				
Feladat	A program klózok száma	Pontosság		
		Pozitív példákon	Negatív példákon	Összesítve
Rectangle	7,8	73,68%	46,15%	59,74%
Sentence	10,2	92,68%	80,00%	86,42%
Shuttle	5,1	99,64%	96,37%	97,93%

3.18. Táblázat. A SPECTRE algoritmus eredményei a bonyolultsági mérték stratégia felhasználásával

A tesztfeladat során kapott eredmények Az IMPUT tanuló algoritmussal, a bonyolultsági mérték stratégia felhasználásával				
Feladat	A program klózok száma	Pontosság		
		Pozitív példákon	Negatív példákon	Összesítve
Rectangle	5,0	100,00%	84,62%	92,21%
Sentence	5,9	73,17%	92,50%	82,71%
Shuttle	4,0	99,64%	97,69%	98,62%

3.19. Táblázat. Az IMPUT algoritmus eredményei a bonyolultsági mérték stratégia felhasználásával

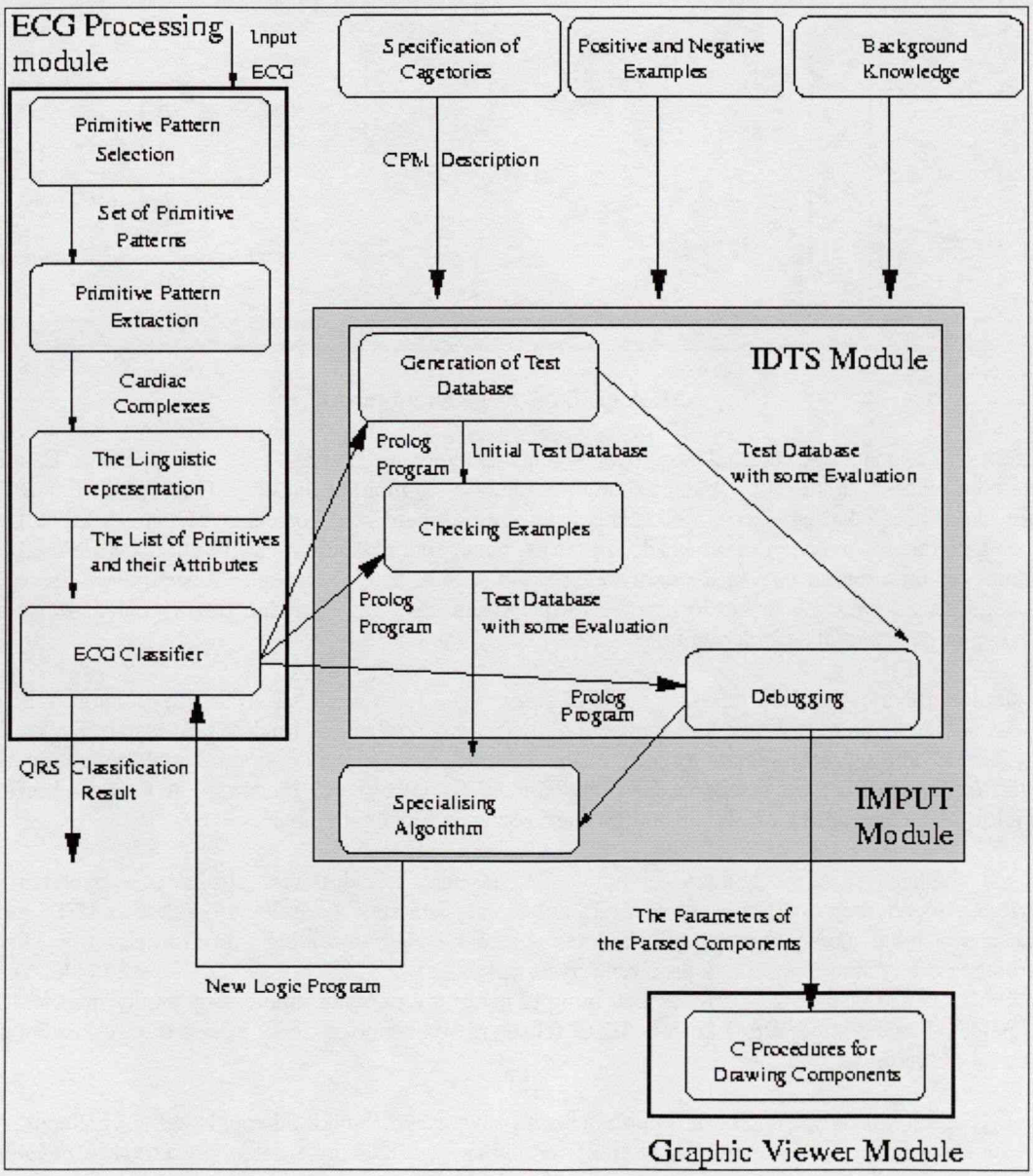
A futtatások során ugyanazt a technikát használtuk, mint Boström [Bos94], azaz a pozitív és a negatív példák halmazát véletlenszerűen két részre bontottuk. Az egyik részt tréning adatként, a másikat teszt adatként használtuk fel. Mind a három feladat esetén tíz véletlen feladatot készítettünk és a pontosság értékeket valamint a klózok számát átlagoltuk. Az eredményeket a 3.17-3.19. táblázatok tartalmazzák.

Az IMPUT algoritmust két különböző stratégiával is teszteltük, az egyik a Prolog stratégia volt, amikor az unfolding transzformációhoz mindig a legbaloldalibb literált választotta az algoritmus, illetve a bonyolultsági mérték stratégiát. Az összehasonlítás kedvéért a SPECTRE algoritmust is lefuttattuk a bonyolultsági mérték stratégiával ugyanezekre a példákra.

Az eredményekből az látható, hogy a bonyolultsági mérték stratégia adja a jobb megoldást mind a klózok számát, mind pedig a pontosságot tekintve. Összehasonlítva az IMPUT és a SPECTRE rendszert észrevehetően kevesebb az IMPUT által tanult hipotézisekben a klózok száma. Az IMPUT tehát tömörebb, kompaktabb hipotéziseket képes tanulni köszönhetően annak a külső tudásnak, ami a CPM teszt adatbázisban valamint a bölcs által begépelt válaszokban van. Ezen felül, több esetben az IMPUT pontosabb eredményt is adott.

3.5. EKG szintaxis tanulásával kiegészített PECG alkalmazás

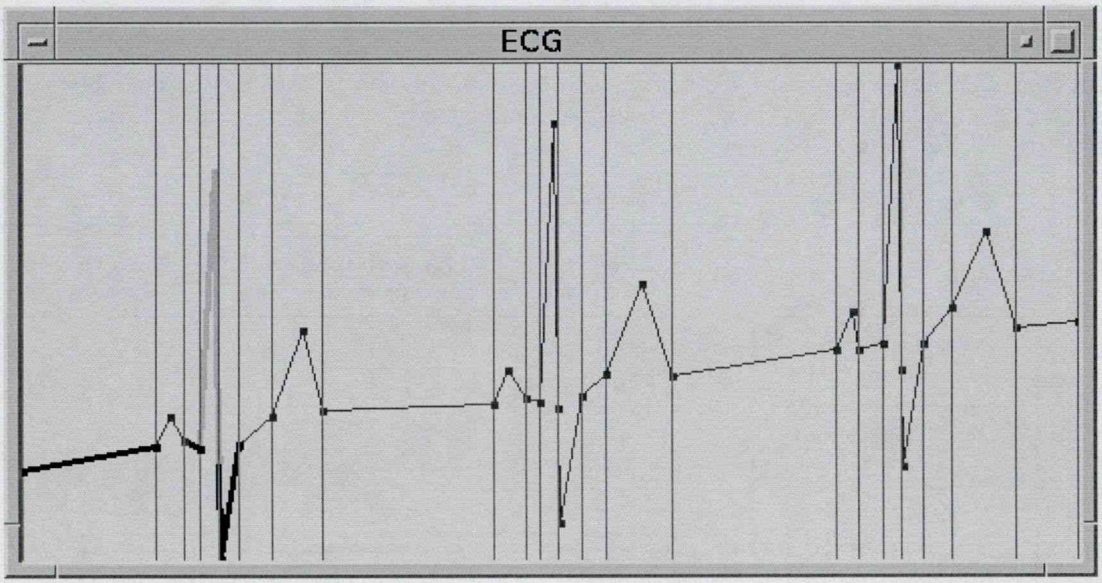
A következőkben a tanulási képességgel kibővített PECG [Kók97] rendszer bemutatására kerül sor. Jelenleg ez a legnagyobb méretű alkalmazása az IMPUT tanuló rendszernek. A tanulással kibővített PECG rendszer egy EKG hullámok osztályozására kifejlesztett program (ez kapta a PECG elnevezést). A rendszer fő moduljai a 3.20. ábrán láthatók. A kibővített PECG (Prolog implementation of ECG) [KókP96] rendszer osztályozza a beérkező EKG hullámokat. Általában a hullámok ismétlődnek, ezért még valamilyen betegség esetén is néhány osztályba sorolhatók, ami a diagnózist jelentősen megkönnyíti. Ha a hullámok csak kisebb mértékben különböznek egymástól, akkor a rendszer egy osztályba helyezi őket. Egy hasonlósági mérték alapján oldható meg a feladat. Egy tíz perces EKG vizsgálat során mintegy 800 hullám keletkezik. Ha ezek között csak néhány rendellenes fordul elő, az megnehezíti az orvos munkáját.



3.20. Ábra. A kibővített PECG rendszer komponensei

A PECG rendszer egy attribútum nyelvtannal adott specifikáción alapul, amely Skordalakis és Papakonstantinou nevéhez fűződik [Skor84], [Skor90]. Ebből Kókai Gabriella és társai készítettek futtatható Prolog rendszert [KókP96]. A végső rendszerbe egy grafikus megjelenítő programot és először az IDTS [Paa94] interaktív algoritmikus hibakereső rendszert integrálták. A felhasználó az egy osztályba sorolt hullámok megtekintése alapján eldöntötte, hogy szükség van-e az osztályozó algoritmus finomítására. Amennyiben igen, akkor a felhasználó segítséget kapott a hibás eljárás lokalizálására a beépített IDTS hibakereső rendszertől.

A PECG rendszerben az EKG hullámokat egy numerikus attribútumokkal kibővített DCG [Per80] nyelvtan írja le. A különböző (egészséges és rendellenes) hullámokhoz a megfelelő nyelvtan elkészítése nehézkes. A nyelvtan fejlesztésében segítséget nyújtó számítógépes eszköz megkönnyítheti ezt a munkát. Emiatt készült el végül a *tanulási funkcióval kibővített PECG rendszer*.



3.21. A grafikus megjelenítő modul

A fejlesztő feladata egy túl-általános (overly-general) nyelvtan elkészítése az első lépésben. Ez a nyelvtan természetesen nemcsak a korrekt, hanem néhány nem megfelelő input hullámot is felismerhet. Ezt az általános nyelvtant általában sokkal egyszerűbb elkészíteni, mint egy hibamentes korrekt nyelvtant. A kibővített PECG inputja ez az általános nyelvtan, mint kezdő hipotézis, háttér tudás, amely az attribútumok kezelésére szolgáló predikátumokat tartalmazza és végül pozitív és negatív példák a nyelvtan specializációjához. A rendszer az IMPUT algoritmust használja a kezdő hipotézis kijavítására úgy, hogy minden pozitív példát felismerjen, de egyetlen egy negatív példát se ismerjen fel.

A tanulási folyamat során a rendszer kérdéseket tesz fel a felhasználónak azzal kapcsolatosan, hogy egyes rész struktúrák amelyeket egyes nemterminálisokból induló rész nyelvtanok írnak le helyesek-e. Ezekre általában nem egyszerű a válasz, ezért található a rendszerben egy grafikus megjelenítő modul, amely meg tudja mutatni a képernyőn az éppen felismert részét az EKG hullámnak. Ez látható a 3.21. ábrán. A képen a sötét (fekete) színnel jelölt szakaszok jelölik az aktuálisan felismert részét az EKG hullámnak.

A kép alapján egy orvos szakértő el tudja dönteni, hogy az adott rész struktúrát a nyelvtan helyesen felismerte-e. Amennyiben az osztályozó program helytelenül ismerte fel a hullámot, akkor az IDTS rendszerrel meghatározza a hibát okozó klózt, majd alkalmazza a már ismert unfolding transzformációt és klóz törlést. Rendszerint ez a folyamat elvezet egy precízebb nyelvtanhoz egy vagy esetleg néhány lépésen belül. A rendszer addig alkalmazza a transzformációs lépéseket, amíg el nem jut egy olyan hipotézisig, amely már nem fedi le a negatív példát. Ezzel a módszerrel korrekt EKG felismerő nyelvtanokat lehet készíteni egy kezdeti általános specifikációból kiindulva.

Az EKG feldolgozás egy a [Skor90] publikációban bevezetett szintaktikus megközelítésen alapul. A *hullám primitív kiválasztás* (primitive pattern selection) modulban az EKG hullámokat primitívekre *dekomponálják* (egyenes szakasz, csúcs, parabola). A következő *primitívek adatainak összegyűjtése* lépésben (primitive pattern extraction) a numerikus attribútumokat határozza meg a rendszer, illetve pontosítja az előző lépés eredményét. A

legnagyobb problémát a zajos csúcsok felismerése jelenti. A *nyelvi reprezentáció* (linguistic representation) modulban az utolsó két lépés eredményét az EKG nyelvtan bemenetét alkotó token sorozattá alakítják. A token egy abc egyik betűje, amely (numerikus) attribútumokkal is ki lehet egészítve.

3.5.1. Az EKG hullámok szintaxisa

Az EKG hullámokat leíró attribútum nyelvtan Skordalakis publikálta [Skor90]. Az input abc, amely a hullámokat alkotó primitíveket tartalmazza a következő volt: $\Sigma = \{K^+, K^-, E, \Pi\}$. Ez a négy szimbólum reprezentálja a pozitív csúcsot (K^+), negatív csúcsot (K^-), az egyenes szakaszt (E) és a parabola szegmenst (Π).

Az itt bemutatott alkalmazásban a fenti attribútum nyelvtannal ekvivalens DCG specifikációt használtak. A DCG specifikációban jól elkülöníthetők a szintaxis leírására szolgáló átírási szabályok és az attribútumok kezelésére szolgáló predikátumok. A kiegészítő grafikus megjelenítő folyamatosan követi a levezetési fát, és eltérő színnel mutatja az aktuális részfának megfelelő EKG hullám darabot.

3.22. Ábra. Egy bemenő EKG jelsorozat

Tegyük fel, hogy a feladat a 3.22. ábrán látható EKG hullám nyelvészeti reprezentációja a következő:

$$E\ K^+\ \Pi\ K^+\ K^-\ E\ K^+\ \Pi\ K^+\ E\ K^+\ K^-\ \Pi\ K^+\ \Pi\ K^+\ \Pi\ K^+\ K^-\ E\ K^+\ \Pi$$

Skordalakis [Skor90] megközelítése szerint minden primitívhez több numerikus attribútum tartozik (pl. a csúcs kezdete és vége, a meredekség jellemzői, a szakaszok kezdete és vége). A PECG rendszer a hullámok helyes felismerése érdekében használja ezeket az attribútumokat. Az itt bemutatásra kerülő példában azonban az EKG nyelvtan szintaktikus részére alkalmazzuk az IMPUT tanuló algoritmust, a numerikus értékek kiszámítása és kezelése a háttér tudáshoz fog tartozni. Az EKG hullám osztályozó program a nyelvészeti reprezentáció alapján ismeri fel a hullámokat.

A kibővített PECG alkalmazásban az IDTS alrendszer teljes szolgáltatásrendszere alkalmazásra került. A szolgáltatásokat három modul biztosítja: A *teszt adatbázis modul* a kezdeti teszt adatbázist készíti el egy CPM teszt specifikáció alapján. A *tesztelés modul* egy kiválasztott predikátum tesztelését végzi, úgy hogy a felhasználó teszt eseteket (reprezentáló elemeket) adhat meg a kiválasztott predikátumhoz tartozó egyes teszt keretekhez. Ezután lefuttatásra kerül a program erre a teszt példára, a felhasználó pedig értékeli a kapott eredményt (korrekt/inkorrekt). A *hibakereső modul* működése során amennyiben már rendelkezésre állnak teszt eredmények bizonyos teszt esetekre, úgy azokat a hibakereső rendszer minden további kérdezés nélkül felhasználja anélkül, hogy a felhasználót „nehéz" kérdésekkel zaklatná.

3.5.2. A tanulási folyamat az IMPUT rendszerrel

A következőkben bemutatásra kerül hogy az integrált rendszer hogyan finomítja a kezdeti EKG nyelvtan .DCG megvalósítását. Példaként vegyük a szív összehúzódási ciklust (cardiac cycle). Minden összehúzódási ciklus egy vagy több hullámból áll, azonban a szív tevékenységének legfontosabb része az ún. QRS komplexum. A QRS komplexumot leíró résznyelvtanon kerül bemutatásra a tanulási folyamat. Az eredeti attribútum nyelvtan szerint a QRS komplexum egy és hét közötti számú csúcsot tartalmaz, amelyek mindegyike megfelel egy QRS mintának. A kezdeti hipotézisből kiindulva a nyelvtannak ez a része kerül majd finomításra úgy, hogy az input példákhoz illeszkedő csúcs sorozat alkossa a QRS komplexumot.

Az IMPUT rendszer bemenő adatai: a kezdeti hipotézis, amely az EKG hullámokat osztályozza, a háttér tudás, a pozitív és a negatív példák. Ez látható a 3.23. ábrán. A rendszer feladata egy olyan hipotézis előállítása, amely kizárólag a pozitív példákat fedi le.

A főbb tevékenységek a következők: a felhasználó elindítja a kibővített PECG rendszert, majd megadja az input adatokat tartalmazó fájl nevét. Minden klózt sorszámmal lát el a rendszer, amely a program futása során megmarad. Az unfolding transzformációval kapott klózokhoz a rendszer egy szám párt rendel, amelyből az első

a klóz eredeti számát mutatja, a második szám pedig a behelyettesített másik klóz sorszámát pl. 3-4 sorszámot kap az a klóz, amelyet a 3 sorszámú klóz egy literáljának helyettesítésével kapunk úgy, hogy a 4-es számú klóz törzsét helyettesítjük a literál helyére.

```
 0: cardiac_cycles --> cardiac_cycle, cardiac_cycles.
 1: cardiac_cycles --> {true}.

 2: cardiac_cycle  --> qrs, non_qrs.

 3: qrs      --> peak,peaks.
 4: peaks    --> peak,peaks.
 5: peaks    --> {true}.

 6: non_qrs --> sr.
 7: non_qrs --> interwave_segment, t, interwave_segment .
 8: non_qrs --> interwave_segment, p, interwave_segment .
 9: non_qrs --> interwave_segment, t, interwave_segment, p, interwave_segment.

10: sr  --> segment, interwave_segment.
11: sr  --> peak, interwave_segment.

12: interwave_segment --> segment, interwave_segment.
13: interwave_segment --> {true}.
14: interwave_segment --> peak, interwave_segment.

15: t  --> t_or_p.
16: p  --> t_or_p.

17: background (t_or_p  --> [p_peak], [n_peak]).
18: background (t_or_p  --> [n_peak], [p_peak]).
19: background (t_or_p  --> [p_peak]).
20: background (t_or_p  --> [n_peak]).
21: background (peak    --> [p_peak]).
22: background (peak    --> [n_peak]).
23: background (segment --> [line]).
24: background (segment --> [par]).

25: positive cardiac_cycles([p_peak,n_peak,line,...],[]).
26: positive cardiac_cycles([n_peak,p_peak,line,...],[]).
27: positive cardiac_cycles([n_peak,p_peak,n_peak,...],[]).
28: positive cardiac_cycles([p_peak,n_peak,p_peak,...],[]).
...
37: negative cardiac_cycles([p_peak,p_peak,line,...],[]).
38: negative cardiac_cycles([n_peak,n_peak,line,p_peak,...],[]).
39: negative cardiac_cycles([p_peak,p_peak,n_peak,...],[]).
40: negative cardiac_cycles([n_peak,n_peak,p_peak,...],[]).
...
```

3.23. Ábra. A PECG tanulási feladat

Először a rendszer a pozitív és a negatív példákat ellenőrzi (a kezdeti hipotézis ebben az esetben mind a pozitív mind pedig a negatív példákat lefedi). A tanuló program először ellenőrzi, hogy a pozitív és negatív példák halmaza különbözik-e, másodszor hogy vannak-e olyan klózok, amelyek nem szükségesek a pozitív példák levezetéséhez (amennyiben az IMPUT úgy van paraméterezve, hogy a legspeciálisabb programot állítsa elő). Ha vannak ilyen klózok, akkor azokat a rendszer törli. Jelen esetben jó néhány klóz törölhető volt (a 7, 8, 9, 11, 12, 14, 15, 16 sorszámú).

Ezután a tanulással kibővített PECG rendszer ellenőrzi a negatív példákat. Ha van olyan negatív példa, amit az aktuális hipotézis lefed, akkor a tanulási funkció aktivizálódik, egyébként a PECG úgy tekinti, hogy sikerült a kívánt korrekt hipotézist meghatároznia.

```
Checking input examples:

 The sets of positive and negative examples are distinct.

 Checking positive examples:

 Clauses were found, that are not needed to cover positive
  examples.
 - clauses were removed from the initial theory.

 The remained clauses are:
 0: cardiac_cycles-->cardiac_cycle,cardiac_cycles
 1: cardiac_cycles-->""
 2: cardiac_cycle-->qrs,non_qrs
 3: qrs-->peak,peaks
 4: peaks-->peak,peaks
 5: peaks-->""
 6: non_qrs-->sr
10: sr-->segment,interwave_segment
13: interwave_segment-->""
```

3.24. Ábra. A PECG tanulási feladat során megmaradt klózok

Az első lépés a tanulás során annak a klóznak a meghatározása a beépített hibakereső programmal, amelyikre az unfolding transzformációt kell alkalmazni. A hibakereső program által feltett kérdésekre *igen* vagy *nem* válaszokat kell adnia a *bölcsnek* (felhasználónak). Ezek a kérdések arra vonatkoznak, hogy egy bizonyos tény benne van-e a megtanulandó program $\mathcal{M}$ szándékolt modelljében (azt tudjuk, hogy a tény a program által generált $\mathcal{M}(P)$ Herbrand-modellben benne van, a rendszer arra kíváncsi, hogy ez ténylegesen is így van-e). Ez egy ún. halmazhoz tartozási kérdés (membership query). Amennyiben a válasz *nem*, úgy a hibakereső rendszer rátalált a hibát okozó klózra. Ekkor elkezdődik az unfolding transzformáció. Ha a válasz igen, akkor a hibakereső program tovább folytatja a levezetési fa postorder bejárását és a kérdezést. Jelen esetben az utolsó kérdés a:

```
 Is it ok [peaks([])] (y/n) y
 Is it ok [peaks([p_peak])] (y/n) y
 Is it ok [qrs([p_peak,p_peak])] (y/n) n
```

A legösszetettebb része a rendszernek az unfolding transzformáció végrehajtása. Feltevés szerint, amennyiben egy negatív példát lefed az aktuális hipotézis, akkor létezik legalább egy olyan klóz, amely felelőssé tehető ezért. A hibakereső algoritmus azt találta, hogy az `peaks --> peak, peaks` klóz a felelős. Ezt a klózt nem lehet törölni, mert részt vesz a pozitív példák levezetésében.

```
 Unfolding at the clause instance:
   3: qrs-->peak([p_peak, p_peak, line,...],
                 [p_peak, line,...]),
            peaks([p_peak, line, p_peak,...],
                  [line, p_peak,...])

 - trying resolvent(s): [3-1] actual minimum is: 12.5101.
 - trying resolvent(s): [3-2] actual minimum is: 0.

The result of the unfolding is:

0: cardiac_cycles-->cardiac_cycle, cardiac_cycles
1: cardiac_cycles-->""
2: cardiac_cycle-->qrs, non_qrs
3-4: qrs-->peak, peak, peaks
3-5: qrs-->peak
 4: peaks-->peak,peaks
 5: peaks-->""
...
```

3.25. Ábra. A PECG feladat az első unfolding transzformáció után

Az új hipotézis nem fedi az első negatív példát, azonban lefedi az összes többit. A hibakereső rendszer ismételt használatával újabb hibás klózt találhatunk. Ezt a tevékenységet folyamatosan ismételjük. A már jól ismert lépésekből most már csak az utolsó kerül bemutatásra.

```
Checking positive examples:

Checking negative examples:

66: cardiac_cycles([n_peak,p_peak,n_peak,...],[]) covered.

 The above theory:
    covers   12 positive samples from 12 (100.00%) and
    fails on 39 negative samples from 40 (97.50%).

 The fact cardiac_cycles([n_peak,p_peak,n_peak,...],[])
  is covered by the theory.
 Starting the false proc. algorithm to determine the basis of the unfolding.

 Unfolding at the clause instance:
 3-4-4-22-22-4-21-4-21-4-22-21-4:
 qrs([n_peak,p_peak,n_peak,...],[line,p_peak,par,...]):-
   C([n_peak,p_peak, ...],n_peak,[p_peak,n_peak,p_peak, ...]),
   C([p_peak,n_peak, ...],p_peak,[n_peak,p_peak,n_peak, ...]),
   C([n_peak,p_peak, ...],n_peak,[p_peak,n_peak,p_peak, ...]),
   C([p_peak,n_peak, ...],p_peak,[n_peak,p_peak,p_peak, ...]),
   C([n_peak,p_peak,p_peak,...],n_peak,[p_peak,p_peak,line,]),
   C([p_peak,p_peak,line,...],p_peak,[p_peak,line,p_peak,...]),
   peak([p_peak,line,p_peak,...],[line,p_peak,par,...]),
   peaks([line,p_peak,par,...][line,p_peak,par,...])

 - trying resolvent(s): [3-4-4-22-22-4-21-4-21-4-22-21-4-7]
   actual minimum is: 3.44203.
 - trying resolvent(s): [3-4-4-22-22-4-21-4-21-4-22-21-4-8]
   actual minimum is: 3.52546.

 The result of the unfolding is:

0: cardiac_cycles --> cardiac_cycle,cardiac_cycles
1: cardiac_cycles --> {true}
2: cardiac_cycle  --> qrs,non_qrs
3-4-4-21-21-4-4-22-4-22-4-21: qrs -->
   [p_peak],[n_peak],[p_peak],[n_peak],[p_peak],
   [n_peak],[p_peak],peaks

3-4-4-21-21-4-4-22-4-22-5: qrs -->
   [p_peak],[n_peak],[p_peak],[n_peak],[p_peak],[n_peak]

3-4-4-21-21-4-4-22-5-21: qrs -->
   [p_peak],[n_peak],[p_peak],[n_peak],[p_peak]

3-4-4-21-21-4-5-22: qrs -->
   [p_peak],[n_peak],[p_peak],[n_peak]

3-4-4-21-21-5: qrs --> [p_peak],[n_peak],[p_peak]

3-4-4-22-22-4-21-4-21-4-22-21-4-22: qrs -->
   [n_peak],[p_peak],[n_peak],[p_peak],[n_peak],
   [p_peak],[n_peak],peaks

3-4-4-22-22-4-21-4-21-4-22-21-5: qrs -->
   [n_peak],[p_peak],[n_peak],[p_peak],[n_peak],[p_peak]

3-4-4-22-22-4-21-4-21-5-22: qrs -->
   [n_peak],[p_peak],[n_peak],[p_peak],[n_peak]

3-4-4-22-22-4-21-5: qrs -->
   [n_peak],[p_peak],[n_peak],[p_peak]

3-4-4-22-22-5: qrs --> [n_peak],peak,[n_peak]
3-4-5-21-22: qrs --> [n_peak],[p_peak]
3-4-5-22-21: qrs --> [p_peak],[n_peak]
3-5: qrs --> peak
```

```
5: peaks --> {true}
6: non_qrs --> sr
10: sr --> segment,interwave_segment
13: interwave_segment --> {true}

 Checking positive examples:

 Checking negative examples:

 The above theory:
    covers   12 positive samples from 12 (100.00%) and
    fails on 40 negative samples from 40 (100.00%).
```

3.26. Ábra. A PECG feladat az első unfolding transzformáció után

Itt a tanulási folyamat véget ér, mert az eredményül kapott hipotézis minden pozitív példát lefed, viszont egyetlen negatív példát sem fed le.

3.6. Hasonló rendszerek és kapcsolódó kutatások

Ebben a fejezetben először összehasonlításra kerül az IMPUT a SPECTRE algoritmussal, a továbbiakban pedig az IMPUT más tanuló rendszerekkel.

Ellentétben más specializációs technikákkal [Bain92], [Ling91] a SPECTRE rendszer nem minimális logikai programot állít elő. Ez azt jelenti, hogy az eredményül kapott hipotézisnek lehetnek további olyan logikai következményei, amelyek nem kapcsolódnak a példákhoz. A SPECTRE, hasonlóan más rendszerekhez [Shap83], [Berg88], [Coh91], [Coh92], [Moo91], [Our90], [Paz91], [Paz93], [Rich91], [Wog91], |Wrob93] nem törekszik arra, hogy minden olyan klózt töröljön, ami nem szükséges a pozitív példák levezetéséhez. Az itt említett rendszerek három egymástól nem teljesen független csoportba oszthatók a felhasznált specializációs módszer alapján: klóz törlés, literál hozzáadás, cél redukálás.

A Wogulis által ismertetett [Wog91] rendszerben az egyetlen specializációs operátor a klóz törlés. A SPECTRE algoritmussal ellentétben ez az algoritmus csak a kezdeti hipotézisben meglévő klózokat képes törölni. Ennek következtében az algoritmus nem tud olyan hipotézist előállítani, amely lefedi az összes pozitív példát, ha a negatív példák levezetésekor használt klózok mind előfordulnak pozitív példa levezetéseiben.

Wrobel [Wrob93] egy inkrementális specializációs technikát alkalmazott (a neve MBR, Minimal Base Revision), amely a klóz törlést és a literál hozzáadást alkalmazta, hogy meggátolja a negatív példák levezetését. A módszernek a lényege nem annyira a program specializáció a pozitív és negatív példák alapján, hanem inkább egy minimális revízió előállítása. A minimális revízió azt jelenti, hogy csak egy minimális klóz halmazt kelljen törölni a kiindulási hipotézisből. Meg kell jegyezni, hogy ilyen értelemben a SPECTRE nem a minimális specializációt állítja elő.

A [Shap83], [Berg88], [Quin90], [Paz91], [Rich91], [Paz93] rendszerekben a klózokat literál hozzáadásával specializálják. A klózok törzséhez hozzáadott literálok azok közül a predikátumokból kerülnek ki, amelyek a kiindulási hipotézisben előfordulnak. Különböző feltételeket alkalmaznak a változókra is, például az új literál minden változója meg kell jelenjen a klózban valahol máshol is [Quin90]. A keresési fa sokkal nagyobb mértékben szélesedik ki az ilyen algoritmusoknál mint a SPECTRE esetében.

A SPECTRE és a specializációs technikák közül a cél redukálást használó rendszerek (ML-SMART [Berg88], ANA-EBL [Coh91], FOCL [Paz91], GRENDEL [Coh92], FOCL-FRONTIER [Paz93]) közötti fő különbség az a mód, ahogyan a alkalmazandó transzformációt megkeresik. Az ANA-EBL, FOCL, GRENDEL és a FOCL-FRONTIER ugyanúgy mint a MIS [Shap83] és a FOIL [Quin90] egy lefedési (pl. szekvenciális lefedés lásd 2.3.1. Fejezet) algoritmust használ, addig a SPECTRE az *„oszd meg és uralkodj"* módszert. Ez azt jelenti, hogy az előbbi algoritmusok a specializációs (keresési) fát újra és újra bejárják (vö. AQ keresési algoritmusok [Mich80]), azonban a SPECTRE a keresési fát csak egyszer járja be (vö. ID3 algoritmus [Quin86]). A fő különbség az ML-SMART és a SPECTRE között az, hogy az ML-SMART csak a legbaloldalibb cél redukcióját veszi figyelembe, addig a SPECTRE egy levezetési stratégia alapján választja ki a redukálandó célt. A levezetési stratégia az algoritmus inputja. A SPECTRE algoritmus tesztelésének eredményei [Bos94] azt mutatták, hogy egy dinamikus levezetési stratégia, amely figyelembe veszi a példákat sokkal pontosabb specializációkat eredményezhet, mint az ML-SMART algoritmusban használt *rögzített* (Prolog) stratégia.

A SPECTRE és az IMPUT algoritmus közötti fő különbség abban van, ahogyan az IMPUT az unfolding transzformációhoz a klózt kiválasztja. A SPECTRE mindig a *cél predikátumra* alkalmazza az unfolding transzformációt. Az IMPUT készítőinek elképzelése az, hogy a sok esetben helyesebb, ha egy a *cél predikátumtól* különböző másik klózra alkalmazzák a transzformációt.

Amikor egy negatív példát lefed az aktuális hipotézis, akkor feltételezés szerint kell legyen egy olyan klóz, amely ezért a viselkedésért felelős. Az IMPUT meg tudja határozni ezt a klózt, és erre alkalmazza az unfolding transzformációt. Ezért az IMPUT egy egyszerre több predikátum revízióját elvégezni képes eszköz, amely egy algoritmikus hibakereső alrendszert tartalmaz a predikátum kiválasztására.

Az IMPUT algoritmushoz hasonlóan a SPECTRE II [Bos95] is használható a cél predikátumtól különböző predikátum definíciójának finomítására. Azonban míg a SPECTRE II-ben ezt a klózt nemdeterminisztikusan választja ki az algoritmus, addig az IMPUT az IDTS hibakereső rendszerrel végzi el, determinisztikusan a kiválasztást. A SPECTRE II fő előnye a korábbi SPECTRE algoritmushoz képest, hogy rekurzív predikátumokat is tud specializálni.

A JIGSAW [Ade95] elmélet revíziós (theory revision) rendszer amennyiben az már meglévő két rendszer integrációja (RUTH [Ade94] és a SPECTRE [Bos94]). Az IMPUT egy interaktív algoritmust használ a hibás klóz meghatározására, amelyre az unfolding transzformációt fogja alkalmazni a későbbiekben, a JIGSAW mélységi iteratív sémát használ annak meghatározására hogy hogyan kell minimális beavatkozással finomítani a kezdeti hipotézist.

A 3. Fejezetben egy új tanuló algoritmus bemutatására került sor, amely logikai programok specializációját tudja elvégezni. Az IMPUT a SPECTRE algoritmus és egy interaktív hibakereső módszer sikeres kombinációja. A megoldásnak egy hátránya, hogy szükség van egy *bölcs* közreműködésére, akinek a megtanulandó rendszer modellje ismeretében igen/nem válaszokat kell adni a program által feltett kérdésekre, ez alapján lehetséges a hibás klóz megtalálása. Ezt ellensúlyozza az, hogy IMPUT-ba integrált IDTS hibakereső rendszer jelentősen tudja csökkenteni a kérdések számát. Ugyancsak hátrányos, hogy a CPM teszt adatbázis használatához egy kezdeti teszt specifikációt kell készíteni, amely az interaktív hibakereső számára egyfajta külső tudásnak tekinthető. Általában azonban van valamilyen előzetes elképzelésünk a megtanulandó program viselkedéséről. Ebből a szempontból a CPM teszt specifikáció úgy tekinthető, mint egy magasabb rendű leírása a programnak, amely nagy hasonlóságot mutat az integritási feltételekkel [DeRa92].

Az IMPUT tanuló algoritmus, amely integrálja a SPECTRE és az IDTS rendszert Prolog nyelven elkészült, azonban vannak olyan feladatok, amelyeket az IMPUT nem képes megoldani. Néhány ezek közül:

```
1.  P = { p(X) },

    E+ = { p(f(a)) },
    E- = { p(g(a)) }.


2.  P = { p(X) },

    E+ = { p((a, b)) },
    E- = { p((a, b, c)) }.


3.  P = { p(X) :- is_list(X) },

    E+ = { p([a]) },
    E- = { p([(a,b)]) }.
```

3.27. Ábra. Feladatok, amelyeket az IMPUT rendszer nem képes megoldani

Annak az oka, hogy az IMPUT rendszer nem képes megoldani a 3.27. ábrán látható feladatokat az, hogy az IMPUT nem képes a klózok változóinak további specializációjára (pl. függvényszimbólumok bevezetése, változók egyezőségének kényszerítése, stb.). Amikor az IMPUT egy egység klózhoz (unit klóz) jut el, amelynek a jobb oldalán nem áll egyetlen literál sem, az algoritmus megáll azzal, hogy: „további unfolding transzformáció már nem lehetséges". Itt véget is ér a specializációs folyamat. Nienhuys-Cheng és Wolf elméleti szempontból tanulmányozták ezt a jelenséget [Chen96] és adtak egy elméletileg teljes megoldást. Mivel azonban csak

egzisztenciális bizonyítást közöltek, ezért a javasolt algoritmus teljes megvalósításának lépései csak részben tisztázottak. Mindazonáltal az IMPUT tanuló algoritmus kibővítése további specializációs operátorokkal egy olyan ígéretes lehetőség, amire szükség esetén érdemes lenne energiát áldozni.

4. A szabályalapú tanuló algoritmusok természetes nyelvi alkalmazásai

Az „ILP2" ESPRIT LTR 20237 projekt fő célkitűzése az volt, hogy a részvevő partnerek megfelelő alkalmazási területeket találjanak a szabályalapú tanuló algoritmusok számára. Az egyik nagy kutatási terület a bioinformatika, amelyen Stephen Muggleton és társai a mai napig élvonalbeli kutatásokat végeznek. Később több partner intézet is bekapcsolódott a kutatásába, például a német GMD[20] bonni Alkalmazott Információtechnológiai Intézete vagy a szlovén Jožef Stefan Institute Intelligens Rendszerek Intézete[21] Ljubljanában. A legfontosabb területek a következők voltak: a fehérjék másodlagos [Moze98], [Mugg92] és harmadlagos szerkezetének predikciója [Turc98], a mutagenitásnak (különböző vegyületek mutáció okozó képességének) [King96], valamint a toxicitásnak (vegyületek sejtpusztító hatásának) a vizsgálata [Srin99]. Az utóbbi két esetben a vegyületek szerkezeti képletének, a kémiai kötéseknek és más adatoknak háttér tudásként való felhasználásával, illetve vegyületek ismert mutagenitásának (toxicitásának) felhasználásával tanuló algoritmusokkal következtettek ismeretlen anyagok mutagenitására (toxicitására). 1997. augusztus 29-e és 1998. november 15-e között rendezték meg a PTE-2[22] (Predictive Toxicology Challange) toxicitás predikció versenyt, amelyben számos tanuló algoritmus vett részt. 30 vegyület toxicitását kellett a megjósolni a kémiai tulajdonságok alapján, majd az eredményeket az állatkísérletek eredményével vetették egybe. Az ILP módszerek kimagaslóan jól szerepeltek [PTE99], a győzelmet a Leuveni Katolikus Egyetem WARMR és a párizsi egyetem[23] MONKEI rendszeréből készített hibrid tanuló algoritmus szerezte meg.

Az mRNS nem-kódoló régióiban található ún. szignál-struktúrák felfedezésére készítettek egy a RIBL tanuló algoritmuson alapuló rendszert Bonnban a GMD Alkalmazott Információtechnológiai Intézet kutatói [Bohn98]. A Progol tanuló algoritmus egy valós alkalmazása volt a neuropeptideket leíró szekvenciák tanulása aminosav szekvenciákból. A Yorki Egyetem kutatói egy generatív nyelvtant próbáltak meg tanulni ILP módszerekkel [MuggT]. A ljubljanai Jožef Stefan Institute Intelligens Rendszerek Intézete és a Leuveni Katolikus Egyetem kutatói vegyületek biológiai lebomlását vizsgálták ILP módszerekkel [Dzer99], majd pedig a szlovén hidrometeorológiai intézettel együtt a folyók vízminőségének biológiai és kémiai paraméterei között kerestek összefüggéseket a TILDE tanuló algoritmussal [Bloc99].

Az egészségügyi alkalmazások között meg kell említeni a rendellenes EKG jelek felismerésének tanulását a 3. Fejezetben ismertetett IMPUT tanuló algoritmussal. A szerzők egy Skordalakis és Papakonstantinou által publikált [Skor84] attribútum nyelvtan Prolog DCG implementációját készítették el, majd az IMPUT tanuló algoritmussal finomították az EKG hullámok szintaxisát leíró program klózokat [Kók96][KókP96]. EKG hullámok folyamatos monitorozására és a rendellenes szívműködés azonnali észlelésére fejlesztettek ki egy alkalmazást Quiniou és társai [Quin01], amelyben az ICL tanuló algoritmust használták fel a rendellenes EKG hullámokat leíró Prolog klózok tanulására. További érdekes alkalmazás Mizoguchi és társai által közölt eljárás szemfenék vizsgálati képek osztályozására zöld hályog diagnosztizálása céljából [Mizo96]. Erre a célra a szerzők egy a tokiói egyetemen kifejlesztett saját tanuló algoritmust készítettek a GKS-t.

A ljubljanai Jožef Stefan Institute Intelligens Rendszerek Intézete, a Leuveni Katolikus Egyetem kutatói és a madridi egyetem közösen egy közúti járműforgalom irányító rendszert vizsgált, amelyben az úttestbe épített szenzorok adataiból tanuló algoritmusok segítségével kritikus forgalmi szituációk felismerését tanulták ILP módszerekkel [Dzer98]. A Yorki Egyetem kutató közúti balesetek adatait dolgozták fel és különlegesen veszélyes útszakaszokat próbáltak meghatározni ILP, valamint attribútum érték tanulási módszerekkel majd a kapott eredményeket összehasonlították [Rob98].

---

[20] Ma már Fraunhofer Gesellschaft, Institut für Angewandte Informationstechnik, Bonn Schloss Birlinghoven http://www.fit.fraunhofer.de/

[21] Jožef Stefan Institute, Department of Intelligent Systems, Ljubljana http://ai.ijs.si/

[22] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PTE/

[23] Univerite Paris-Sud, Laboratoire de Recherche en Informatique.

Ugyancsak említésre érdemes Dolšak és Muggleton [Dol92] alkalmazása, amelyben a GOLEM tanuló algoritmust 3 dimenziós testek szilárdságtani vizsgálatához fejlesztettek ki egy véges elem felbontást végző rendszert. Később Dolšak és Džeroski [Dzer92] a FOIL és az mFOIL rendszerrel megismételték ugyanezt.

Erika Van Baelen és Luc De Raedt tanulmánya zongoraművészek játékának tanulmányozásáról, amelyben a CLAUDIEN tanuló rendszert használták fel arra, hogy egy-egy művész játékára jellemző szabályokat tanuljon meg az előadás MIDI formában rögzített adataiból [Eri96]. Nico Jacobs és Hendrik Blockeel a WARMR tanuló rendszert használta fel arra, hogy Unix shell log fájlok tartalmát feldolgozza és az egyes felhasználók viselkedésére jellemző mintákat tanuljon [Jaco01].

4.1. Fontosabb természetes nyelvi problémák

A szabályalapú tanuló algoritmusok alkalmazásainak egy igen jelentős csoportját alkotják a természetes nyelvhez kapcsolódó feladatok. Az első fontos probléma a beszédfelismerés – ez teremti meg a lehetőséget a felhasználó és a számítógép szóbeli kommunikációjára. A beszédfelismerés eredménye a kimondott szöveg írott alakban, ennek felhasználásával történik meg a további feldolgozás. Az írott szövegből kiindulva a feldolgozás lépései először a mondatokra és szavakra bontás, majd a szavak morfo-szintaktikai elemzése. A szavak morfo-szintaktikai elemzése feltétlenül szükséges a feldolgozás következő lépéséhez a szintaktikai elemzéshez. Egy közbenső feladat lehet a *szófaji egyértelműsítés* (angolul part-of-speech tagging), amely a több lehetséges morfológiai címkével rendelkező szavak lehetséges jelentéseinek számát csökkenti, – ideális esetben a szövegkörnyezetnek megfelelő *egyetlen* helyes címkét adja meg. Az egyértelműsített szöveg szintaktikai elemzése ezután várhatóan gyorsabban történik meg. A szintaktikai elemzés után a szemantikus annotáció következik, amelynek eredményeként végül lehetővé válhat a szöveg megértése.

A számítógépes szövegfeldolgozásban további kutatási feladatok is vannak: az Internet elterjedésének következtében igen fontos feladat a nagyszámú dokumentum tartalmi osztályozása. A megoldás során előre adott véges számú osztályba kell sorolni a Web lapokat egy – a lehetőség szerint – gyors algoritmussal.

A fenti ismertetett feladatok közül nem mindegyik esetében lehetséges tanuló algoritmusok alkalmazása. Sok esetben használnak fel statisztikai módszereket is. A szerző tevékenysége a szabályalapú rendszerekre terjed ki, ezért a továbbiakban olyan feladatok ismertetésére kerül sor, amelyek megoldásában lehetőség nyílik szabályalapú tanuló algoritmusok használatára.

A sok ragot és jelet tartalmazó ún. *agglutináló* (toldalékoló) nyelvek[24] esetében fontos feladat a szavak morfo-szintaktikai elemzése. Ennek során a szavakhoz hozzá kell rendelni a különféle toldalékokból származó attribútumok halmazát (nem, eset, szám, személy, igeidő, fokszám, birtokos személyrag, stb.). A szlovén nyelv – hasonlóan a magyarhoz – számos toldalékkal rendelkezik (a névszóknak neme is van, az esetragok száma viszont kevesebb a prepozíciók használata miatt). A ljubljanai Jožef Stefan Institute Intelligens Rendszerek Intézete és a Yorki Egyetem kutatói közösen kidolgoztak egy a Progol tanuló rendszeren alapuló morfológiai elemző rendszert [Cuss99], majd egy másik rendszert, amely a CLOG tanuló programot használta [Man98]. Az angol nyelv nem tartozik az agglutináló nyelvek közé, ennek ellenére egyes esetekben lehetőség kínálkozik tanuló algoritmusok használatára. A Yorki Egyetem munkatársai az angol múlt idő képzését vizsgálták figyelemmel a számos rendhagyó esetre. A Progol ILP tanuló algoritmust egy genetikai algoritmussal kombinálva használták fel a múlt idő jelét felismerő szabályrendszer tanulására [Kaz98].

A szerző három téma kutatásában vett részt, bár tevékenységének súlypontja főként egy témára, – a szófaji egyértelműsítésre összpontosult. Ez a három terület a következő:

- természetes nyelvi interfészek
- fonéma felismerés
- szófaji egyértelműsítés

Az ember-számítógép kapcsolat minősége fontos szempont az alkalmazások használhatóságának értékelésekor. Az ember szempontjából a legegyszerűbb interfész a természetes nyelvű, ami azt jelenti, hogy a felhasználónak a legkevésbé kell alkalmazkodnia a géphez, egy adott doménen belül (ahol adottak a lehetséges

[24] Ilyenek a török, a szláv vagy a finn-ugor nyelvek, például a magyar is.

tevékenységek és az objektumok, amelyekre a cselekvések irányulhatnak) „*szabadon*" természetes nyelvi parancsokkal végeztethet el műveleteket. A szerző és munkatársai egy attribútum alapú természetes nyelvi interfészt készítettek, amelynek segítségével síkgeometriai szerkesztések elvégzésére lehetett a számítógépet utasítani.

A számítógépes nyelvfeldolgozás a beszédfelismeréssel kezdődik. A beszéd a számítógépbe digitalizált hang formájában kerül be, amelyet egy alkalmas szoftvernek írott szöveggé kell alakítania. A beszéd alapegységeit a *fonémák* alkotják. Általában a magánhangzók egy-egy fonémának felelnek meg, a mássalhangzók, pedig néhány fonéma szekvenciájának. A beszédfelismerés első és igen fontos feladata a szegmentálás – a folyamatosan érkező hanghullám felbontása fonémákra, majd ezután következik az egyes fonémák felismerése.

Az (írott) természetes nyelvi feldolgozás a tokenekre[25] bontással kezdődik, amit a szavak morfo-szintaktikai (vagy egyszerűbben mondva *szófaji*) elemzése követ. A szófaji elemzés azt jelenti, hogy minden egyes szóhoz hozzárendelünk egy attribútum halmazt mint címkét a felismert különféle jelek és ragok alapján. Ez egy nyelvészeti feladat, amelyet egy alkalmas szoftver végezhet el, egy megfelelően nagy méretű lexikon támogatásával. A szófaji egyértelműsítésre azért van szükség, mert a természetes nyelvekben sok szóhoz több különböző szófaji címke is rendelhető. Ezek a többjelentésű szavak pl. *vár, ég, hatnak, tárolt, jobb.*[26] A következő feldolgozási lépés – a szintaktikus elemzés – előtt szükséges az egyértelműsítés elvégzése, ugyanis kiderült a magyar nyelv vizsgálatából, hogy az előforduló szavak közül átlagosan minden másodiknak van több lehetséges címkéje. Ezeket a bizonytalanságokat a szintaktikus elemzés szintjén sokkal nehezebb lenne kezelni. A tanuló algoritmusok ebben az esetben a szavak egyszerű környezetei alapján tanulnak egyértelműsítési szabályokat. Ezzel e közbenső lépéssel hatékonyabbá és gyorsabbá lehet tenni a szintaktikus elemzést.

A szakirodalomban kételyek merültek fel azzal kapcsolatosan, hogy mekkora pontosságot érhetnek el egyáltalán az egyértelműsítő programok. A bizonytalanságok egy része például nem oldható fel csak egy mondatbeli környezet alapján, sőt esetleg a teljes szöveg sem elegendő ehhez (további háttér ismeretek szükségesek). Az elvben feloldható bizonytalanságok esetében pedig felmerül az a kérdés, hogy mekkora tréning halmaz figyelembevétele alapján várható viszonylag pontos eredmény. A kísérletek azt mutatták, hogy a magyar nyelvben megfigyelhető bizonytalanság típusok száma igen nagy, ezért még a millió szó nagyságrendű tréning adatbázis sem biztosan elegendő.

A következőkben a fenti három probléma részletes ismertetése következik a szerző és munkatársai által publikált kutatási eredmények alapján.

4.1.1. Fonéma felismerés

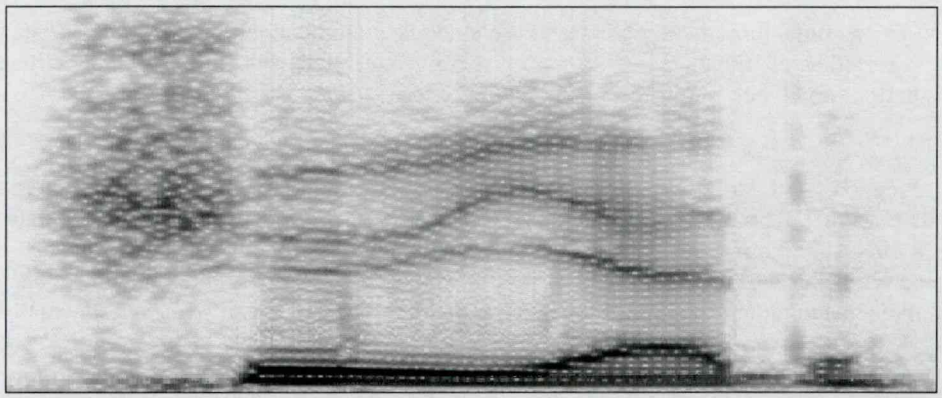
A fonéma felismerésről szóló fejezetek teljes, angol nyelvű szövege megjelent az [Ale97] publikációban.

A jelenlegi beszédfelismerő rendszerek alapvetően két nagy csoportba sorolhatók: a tudás alapú és a sztochasztikus megközelítést alkalmazó rendszerek csoportjába. Az előbbi csoportba tartozó rendszerekben szabályok egy halmazát alkalmazzák a beszéd akusztikus-fonetikai dekódolására. A sztochasztikus rendszerek általában, HMM (Hidden Markov Model) technikát [Rab93] vagy neurális hálózatokat [Mam95] alkalmaznak. Mindkét esetben a felismerési folyamat az előfeldolgozással kezdődik, amely a digitalizált beszédjeleket átalakítja idő-frekvencia reprezentációra. Ez azt mutatja meg, hogy az energia hogyan oszlik el az idő és a frekvencia függvényében a digitális jel mentén. A leggyakrabban használt reprezentáció a *spektogram*, amelyben a jelet rövidebb szegmensekre bontják fel, majd ezekre a szegmensekre alkalmazzák a DFT (Discrete Fourier Transformation) műveletet. Az 4.1. ábra egy tipikus spektogramot mutat be, amely a **süllyed** szó spektogramját jeleníti meg.

---

[25] A token a mondat kisebb elemi egységeit jelöli. Leggyakrabban a tokenek szavak, de természetesen más fontos objektumok is előfordulnak közöttük, mint pl. számok, írásjelek, betűszavak továbbá olyan különleges objektumok is, mint gépjárműrendszám, adószám, bankszámlaszám, iktatószám, sport eredmény, e-mail cím, web cím.

[26] A *vár* és az *ég* szavak lehetnek egyrészt főnevek, másrészt igék. A *hatnak* szó lehet ige jelen idő, többes szám harmadik személyben, illetve a hat számnév részeshatározós vagy birtokos esete. A *tárolt* szó lehet múlt idejű ige vagy befejezett melléknévi igenév. A *jobb* lehet a jó melléknév középfokú alakja, de lehet alapfokú melléknév is. A szófaji címkékben rögzített attribútumok nem terjednek ki a szemantikai többértelműségekre egy szófajon belül (pl. *daru* főnév), munkagép vagy egy madár neve. Ezeknek a feloldása egy további feladat.



4.1. Ábra. A süllyed szó spektogramja

A sztochasztikus rendszerekben is ilyen vagy hasonló reprezentációt használnak a fonémák felismerésére olyan módon, hogy egy eloszlást számítanak ki a beérkező jelből, majd összehasonlítják egy a felismerendő jelhez tartozó tárolt (tanulási folyamattal meghatározott) eloszlással. Általában távolság metrikát használnak arra, hogy két spektogram szegmens (a két spektogram egy-egy oszlopának) távolságát kiszámítsák. A két spektogram egymáshoz illesztését az idő tengely mentén dinamikus programozási módszerekkel (DTW, Dynamic Time Warping) végzik, vagy állapot átmenetekkel szimulálják (HMM alkalmazása esetén). Nagy hátránya a sztochasztikus megközelítésnek, hogy a különböző fonéma típusok különbözőképpen jelentkeznek a spektogramokon, például a magánhangzók esetén a spektrális maximum helyek hordozzák a lényegi információt, míg plozív (zárhangok: **b** vagy **p**) hangoknál a spektrum időbeli változása a fontos [Lam93]. Mivel ezek a rendszerek általános módszereket használnak a jel minden részéhez, nem tudnak a különböző fonémák speciális tulajdonságaira koncentrálni.

A tudás-alapú rendszerekben fonetikus jellemzőket [Hueb94], [Biter95] határoznak meg a spektogramból. A jellemző adatokat a spektogramban akusztikus események, jelek felismerésével lehet meghatározni. Az előfeldolgozó modul outputja egy idő-fonetikus jellemző leképezés, amely az észlelt fonetikus jellemzőket mutatja egy adott pillanatban. A következő lépés az akusztikus jellemzők sorozatának konvertálása fonéma sorozatokká. A konverzió közben fonotaktikus szabályok alkalmazására is sor kerülhet, például ha egy erős [s][27] fonémát egy [d] fonéma követ, amelynek a zöngéssége csekély, akkor a [d] fonémát át kell javítani [t]-re, mert a fonotaktikus szabályok szerint egy [s]-t sosem követhet [d].

4.1.2. Természetes nyelvi interfészek

A természetes nyelvi interfészekről szóló fejezetek teljes, angol nyelvű szövege megjelent az [Ale90] és az [AleA90] publikációkban.

A természetes nyelvi interfészek a természetesnyelv-feldolgozás egyik legkézenfekvőbb alkalmazásai lehetnek [Schr88]. Az üzleti világban a piaci sikerük nyilvánvaló. Az információ feldolgozás és adatbázis kezelés számos területén a természetes nyelvi interfészek kényelmes eszközt jelentenek az információ manipulálására. Bár az eddig kifejlesztett interfészek potenciális vetélytársai az információ-elérés közismert módjainak, különösen a nem-szakember felhasználók számára (vö. menük, ikonok, formális lekérdező nyelvek, mint az SQL), jelentősebb alkalmazásaikat akadályozzák az elkészítés fázisában fellépő különböző nehézségek. Annak ellenére, hogy a felhasználók számára a jelenleg használt interfészek használata nagyobb erőfeszítést igényel, mivel egy tanulási folyamat során a felhasználónak meg kell ismernie a felhasználói interfészt. Ideális esetben egy természetes nyelvi interfész lehetővé tenné a felhasználónak, hogy csak leüljön egy terminál elé és anyanyelvén, folyamatos mondatokkal utasítsa a számítógépet.

Jelenleg ilyen könnyen kezelhető számítógépes programok csak egy jelentősen korlátozott doménre készülnek. Azonban a természetes nyelvi interfészekkel kapcsolatos kutatások eredményeképpen egyre nagyobb területre kiterjedő résznyelvekkel dolgoznak, ami elvezethet a természetes nyelvek komplexitásának jobb megismeréséhez, egyre tökéletesebb interfészek kidolgozásához.

---

[27] A magyar kiejtett *sz* hangot jelenti ez a fonetikus jelölés.

Mind elméleti, mind gyakorlati szempontból alapvetően két éles határvonallal elválasztható típusú természetes nyelvi interfész létezik. Egyik fajtájuk az információ visszakereső programok (adatbázis lekérdező, szakértői rendszerek), amelyek fogalmilag egyszerű adatokhoz férnek hozzá, logikai, aritmetikai és halmaz műveletek segítségével állítják elő a kívánt választ a kérdésekre. A szemantika ebben az esetben egyszerű, előre beégetett. Egyes rendszerekben a szótár szinonim jelentésű szavakkal bővíthető, pl. JAKE[28]. A másik típus a problémamegoldó feladatokra kifejlesztett felhasználói interfész. Ez utóbbi nevezhető esetleg *programozható* természetes nyelvi interfésznek vagy természetes nyelv megértő interfésznek.

Az egyszerű interfészek minimális szintaxissal (vagy szintaxis nélkül) működnek. A parancsok elemzése *kulcsszavak* keresését jelenti, amelyek szemantikus jelentéssel rendelkeznek. A végrehajtás a kulcsszavak által kijelölt szemantikus tevékenységek elvégzését jelenti a kulcsszavak előfordulási sorrendjében. A természetes nyelv megértő interfészek esetében a szemantika az alkalmazott résznyelvet definiáló nyelvtanon alapul. Az utasításként megadott mondat struktúráját a rendszer elemzi, felhasználja a beépített tudásbázist és a feladat egy belső reprezentációját számítja ki. Ez a belső reprezentáció teszi végrehajthatóvá a mondatot. A szerző és munkatársai egy síkgeometriai szerkesztések elvégzésére szolgáló, természetes nyelvet értő, felhasználói interfészt fejlesztettek ki a THALES-t, amely komplex input szintaxissal és egy erre épülő szemantikával rendelkezik.

A korlátozott nyelvi doménre kiterjedő természetes nyelvi interfészek legfontosabb problémája a megfelelő szintaxis és az erre épülő szemantikus tevékenységek kidolgozása. A Kunth által először bevezetett attribútum nyelvtan konstrukció [Knu68] alkalmas eszköz lehet a természetes nyelvi interfészek formális (pontos) definiálására. A THALES rendszer szintaktikai elemző modulja és a szemantikus memóriája attribútum nyelvtanok alkalmazására, továbbá egy attribútum nyelvtan alapú fordítóprogram generáló rendszerre, a PROF-LP-re [Gyim88][Toc88] épül, amelynek a kifejlesztésében a szerző is részt vett, és amelynek segítségével az attribútum nyelvtan specifikációk futtatható program kóddá voltak alakíthatók.

A később részletesen is ismertetésre kerülő rendszer attribútum nyelvtan specifikációit a közreműködő kutatók manuális munkával készítették. Azonban felmerül az a kérdés, hogy nem lehetne-e tanuló algoritmusokkal segíteni legalább a szintaktikus szabályok meghatározását. Mooney [Moo96] ismertetett egy algoritmust, amely az általa készített CHILLIN tanuló rendszer (lásd 2.3.3. Fejezet) felhasználásával tanult szintaktikai szabályokat egy természetes nyelvi elemző rendszer a CHILL számára. A rendszer egy alkalmazása egy földrajzi információs rendszer természetes nyelvi interfésze volt.

4.1.3. Szófaji egyértelműsítés

A szófaji egyértelműsítésről szóló fejezetek teljes, angol nyelvű szövege megjelent az [AleW99] és a [Hor99] publikációkban.

A szófaji egyértelműsítés problémáját a kilencvenes évek elején tanulmányozták intenzíven először, amikor nagy szöveg adatbázisok számítógépes feldolgozása kezdődött el. Nyilvánvalóvá vált, hogy egyes nyelvek, különösen az *angol,* jelentős számban tartalmaz többértelmű szavakat. A probléma megoldására kezdetben kizárólag valószínűség-számítási alapokon nyugvó módszereket alkalmaztak (Bayes hálók, Rejtett Markov Modell), később azonban megjelentek például a szimbolikus tanulási módszerek is. A szófaji egyértelműsítő programot az angol szaknyelvben **tagger**-nek nevezik, a dolgozatban a *tagger* és az *egyértelműsítő program* szavak egymás szinonimáiként fordulnak elő.

Az egyik első ILP tanuló rendszeren alapuló egyértelműsítő programot Cussens ismertette [Cuss97]. A Progol [Mugg95] tanuló rendszert felhasználva tanult angol egyértelműsítési szabályokat. Azokban az esetekben, amikor a Progollal meghatározott szabályok nem tudtak dönteni, akkor a legvalószínűbb morfológiai kategóriát rendelte hozzá a program az adott többértelmű szóhoz. A Progol tanuló rendszert egy olyan korpuszon tanították, amely 3 millió szót tartalmazott és ún. *"removable"* szabályokat (olyan szabályokat, amelyek arra vonatkoznak, hogy a tekintett szó címkéje mi nem lehet) tanultak a többértelmű szavakra: mind ahányszor egy tanult szabály alkalmazható volt egy szó előfordulására a szövegben, a szabályban szereplő címkét eltávolították a lehetséges címkék halmazából. Egy nagyon egyszerű nyelvészeti háttértudás felhasználásával a generált tagger 96,4%

[28] JAKE: The Application-Independent Natural Language User Interface, made by English Knowledge Systems Inc., Scotts Valley, California, USA, 1988.

szavankénti pontosságot[29] ért el.

Daelemans és társai [Dael98] ismertették az MBT (Memory-Based Tagger) rendszert, amelyet különböző nyelvekre alkalmaztak. Az angol nyelvű Wall Street Journal korpusz egy 2 millió szavas részletén tanították az algoritmust, amely 96,4% szavankénti pontosságot ért el. Más nyelvek esetén a következő eredményeket kapták: holland 95,7%, spanyol 97,8% és cseh 93,6%.

Eineborg és Lindberg egy svéd nyelvre alkalmazható egyértelműsítő rendszert publikáltak [Eine98]. Hasonló megközelítést alkalmaztak, mint Cussens [Cuss97], ők is a Progol rendszert használták és *"removable"* szabályokat tanultak a többértelmű szavakra. Az 1 millió szót tartalmazó svéd UMEÅ korpuszt használták fel és 97% szavankénti pontosságot értek el. Később rendszerüket nyelvészeti háttértudás hozzáadásával tökéletesítették [Eine99].

A tanult egyértelműsítő programokat kombinálva még pontosabb algoritmusokat kaphatunk. Halteren és társai [Halt98] négy jól ismert egyértelműsítési szabály generátor rendszer néhány lehetséges kombinációját vizsgálták. Megmutatták, hogy az általuk konstruált mindegyik kombinált tagger algoritmus pontossága jobbnak bizonyult, mint a legjobb komponensük pontossága. A legjobb kombinált taggert *páronként-szavazónak*[30] nevezték el. Ez 97,2% szavankénti pontosságot ért el az 1 millió szavas LOB korpusz felhasználásával.

A Brill tagger generátor rendszer[31] egy alkalmazását mutatta be a Stockholmban tanuló Megyesi Beáta diplomamunkájában a magyar nyelvre [Megy98]. A mintegy százezer szavas [TELRI] korpuszon tanítva 87,5% szavankénti pontosságot ért el. A tapasztalt eredmények azt mutatják, hogy Brill módszere nehézkesen alkalmazható olyan nyelvekre, amelyek tulajdonságai nagyobb mértékben különböznek az angol nyelv tulajdonságaitól.

4.2. Attribútum nyelvtanok használata a természetesnyelv-feldolgozásban

Az attribútumokat Knuth vezette be 1968-ban [Knu68]. Az elgondolása – miszerint a generatív (környezet-független) nyelvtanokat attribútumokkal bővíti ki, az átírási szabályokhoz attribútumokon értelmezett szemantikus tevékenységeket rendel és az attribútumokon végzett számítások eredménye kihathat magára az elemzésre is – annyira sikeresnek bizonyult, hogy az informatikában a mai napig elterjedten használják fordítóprogramok formális specifikációjának az elkészítésére.

4.1. Definíció: Attribútum nyelvtan [Alb91]: egy $AG = (G, SD, AD, R, C)$ öt komponensű struktúra, ahol

(1) $G = (V_N, V_T, P, S)$ egy környezet-független nyelvtan, V_N és V_T a nemterminálisok és a terminális egy véges halmaza, P az átírási szabályok halmaza, S pedig a start szimbólum. $V = V_N \cup V_T$, $V_N \cap V_T = \varnothing$; a terminális és a nemterminális szimbólumok együtt egy V szótárat (*vocabulary*) alkotnak, $S \in V_N$ továbbá egy $p \in P$ szabály a következő alakú: $p: X_{p0} \rightarrow X_{p1} X_{p2} \ldots X_{pn_p}$, ahol $n_p \geq 0$, $X_{p0} \in V_N$, és $X_{pk} \in V$, $1 \leq k \leq n_p$. A G nyelvtan redukált abban az értelemben, hogy minden nemterminális szimbólum elérhető az S startszimbólumból és minden nemterminális átírható egy szimbólumsorozatra, amelyben már csak terminálisok vannak.

(2) $SD = (TYPE\text{-}SET, FUNC\text{-}SET)$, az ún. szemantikus domén. A *TYPE-SET* típusok egy véges halmaza, a *FUCT-SET* pedig $type_1 \times type_2 \times \ldots \times type_n \rightarrow type_0$ típusú teljes függvények, ahol $n \geq 0$ és $type_i \in TYPE\text{-}SET$, $(0 \leq i \leq n)$.

(3) $AD = (A, I, S, TYPE)$ az attribútumok definíciói. Minden $X \in V$ szimbólum rendelkezik attribútumok egy $A(X)$ halmazával, amit két egymástól idegen halmazra lehet bontani: az $I(X)$ és az $S(X)$-re, az örökölt (*inherited*) és a szintetizált (*synthesized*) attribútumok halmazára ebben a sorrendben. Az összes attribútum halmaza $A = \cup_{X \in V} A(X)$. A különböző szimbólumokhoz tartozó attribútumok különbözőek, azaz $A(X) \cap A(Y) = \varnothing$, ha $X \neq Y$.

---

[29] Per-word accuracy, amelyet a következőképpen lehet definiálni: a teszt példán lefuttatott program által helyesen meghatározott szófaji kategóriák száma (az egyértelmű szavakat is beleszámítva) osztva az összes szó számával. A kiszámítás során az írásjeleket figyelmen kívül hagyjuk.

[30] Pairwise-voting

[31] Eric Brill honlapjáról letölthető http://www.cs.jhu.edu/~brill/

Egy X szimbólum a attribútumát $X.a$-val jelölik. Minden $a \in A$ attribútumra $TYPE(a) \in TYPE\text{-}SET$, az a attribútum lehetséges értékeinek halmaza.

(4) R az attribútumok kiértékelési szabályait jelöli. Minden $p \in P$ átírási szabályhoz tartozik egy $R(p)$ véges halmaz, amely az adott szabályhoz tartozó szemantikus egyenleteket tartalmazza $R = \cup_{p \in P} R(p)$. Egy $p \in P$ átírási szabály $p: X_{p0} \rightarrow X_{p1} X_{p2} \ldots X_{pn_p}$ rendelkezik egy (a, p, k) attribútum előfordulással, amennyiben $a \in A(X_{pk})$. Egy $p \in P$ átírási szabályban előforduló összes attribútum előfordulás jele $AO(p)$ (*attribute occurrence*). Ez a halmaz két egymástól idegen részre bontható: a $DO(p)$ és az $UO(p)$-re, a definiáló előfordulások (*defining occurrence*) és a felhasznált előfordulások (*used occurrence*) halmazára:

$$DO(p) = \{(s, p, 0) \mid s \in S(X_{p0})\} \cup \{(i, p, k) \mid i \in I(X_{pk}), 1 \le k \le n_p\}$$

$$UO(p) = \{(i, p, 0) \mid s \in I(X_{p0})\} \cup \{(s, p, k) \mid i \in S(X_{pk}), 1 \le k \le n_p\}$$

Az $R(p)$ halmazban lévő szemantikus egyenletek azt írják le, hogy hogyan kell a $DO(p)$ attribútumokat kiszámítani más $AO(p)$ halmazbeli attribútum előfordulásokból. A kiértékelő függvény a következő alakú:

$$(a, p, k) := f((a_1, p, k_1), (a_2, p, k_2), \ldots, (a_m, p, k_m))$$

ahol az $(a, p, k) \in DO(p), f: TYPE(a_1) \times TYPE(a_2) \times \ldots \times TYPE(a_m) \rightarrow TYPE(a), f \in FUNCT\text{-}SET$ és $(a_i, p, k_i) \in AO(p)$ minden $1 \le i \le m$. Ebben az esetben azt mondjuk, hogy az (a, p, k) attribútum előfordulás függ az (a_i, p, k_i) attribútum előfordulásoktól minden $1 \le i \le m$. Egy attribútum nyelvtan normál alakú, ha egy további feltétel is teljesül, nevezetesen: $(a_i, p, k_i) \in UO(p)$ minden $1 \le i \le m$. Tetszőleges attribútum nyelvtan függvény helyettesítésekkel normál alakra hozható, ezért feltehetjük, hogy a továbbiakban minden attribútum nyelvtan ilyen.

(5) C a szemantikus feltételek halmazát jelöli. Minden $p \in P$ átírási szabályhoz tartozik egy $C(p)$ véges halmaz, amely az adott szabályhoz tartozó szemantikus egyenleteket tartalmazza $C = \cup_{p \in P} C(p)$. Ezek a feltételek (valójában predikátumok) a következő alakúak:

$$f((a_1, p, k_1), (a_2, p, k_2), \ldots, (a_m, p, k_m))$$

ahol $f: TYPE(a_1) \times TYPE(a_2) \times \ldots \times TYPE(a_m) \rightarrow \{true, false\}, f \in FUNCT\text{-}SET$ és $(a_i, p, k_i) \in AO(p)$ minden $1 \le i \le m$. A szemantikus feltételek befolyásolják az elemzést. Egy G által generált mondatot az AG attribútum nyelvtan csak akkor generál, ha minden a levezetéskor kiszámított szemantikus feltétel igaz.

Kezdetben mind a formális nyelvi elemzésben, és ennek következtében az attribútum nyelvtanok kiértékelő eljárásaiban is nagy hangsúlyt helyeztek a determinisztikus viselkedésre. Amennyiben a gyakorlatban is működő rendszereket kívántak építeni, úgy a számítógépek mindenkori számítási kapacitását figyelembe kellett venni. Egy bonyolult, visszalépéseket is lehetővé tevő módszer megvalósítása részben szoftver technikai okokból, de főként hardver okok miatt csak később vált lehetővé. Egy determinisztikus algoritmus futási ideje (művelet igénye) jóval alacsonyabb ugyanazon az inputon, mint egy nemdeterminisztikusé.

4.2. Definíció: S-attribútumos attribútum nyelvtan [Knu68], [Alb91]: egy attribútum nyelvtant S-attribútumosnak nevezünk, ha nem tartalmaz örökölt attribútumokat (azaz csak szintetizáltat attribútumokkal rendelkezik). Tetszőleges $X \in V$ szimbólumra az $I(X) = \varnothing$.

4.3. Definíció: L-attribútumos attribútum nyelvtan [Alb91]: egy attribútum nyelvtant L-attribútumosnak nevezünk, ha bármely $p \in P$ átírási szabályban $p: X_{p0} \rightarrow X_{p1} X_{p2} \ldots X_{pn_p}$ tetszőleges X_{pi} $(1 \le i \le n_p)$ szimbólum örökölt attribútuma csak az $I(X_{p0}) \cup_{1 \le j < i} S(X_j)$ attribútum halmaztól függhet.

Az attribútum nyelvtanok esetében nemcsak az elemzési fa előállítása szükséges egy adott inputra, hanem a fában található összes attribútum példányt is ki kell értékelni. A kiértékelés módját legnagyobb mértékben az attribútum példányok közötti lehetséges függések határozzák meg. Tekintve, hogy egy elemző eljárás működik minden input feladatra, ezért azt úgy kell megkonstruálni, hogy az tetszőleges, a környezet-független nyelvtannak eleget tevő input esetén ki tudja számítani az elemzési fában megjelenő attribútumokat. Minél bonyolultabb a lehetséges függések rendszere, annál bonyolultabb kiértékelési eljárás szükséges.

Az attribútum nyelvtanok különleges osztályát alkotják azok az attribútum nyelvtanok, amelyekben az attribútum kiértékelés és az elemzés egyszerre történhet. Az L-attribútumos nyelvtanok alkotják ezt az osztályt,

amelynek kitüntetett szerepe van a gyakorlati alkalmazásokban (4.3. definíció). Egy ennél is egyszerűbb osztály az S-attribútumos nyelvtanok osztálya, amelyben a kiértékelés még sokkal egyszerűbb. Bonyolultabb esetben az elemzési fa előállítása után több menetben (esetleg különböző irányokban) kell az elemzési fát bejárni egyre újabb és újabb attribútumokat kiszámítva a meglévőkből, hogy a végén előálljon az összes kiszámított attribútumot tartalmazó szintaxis fa.

Természetesen a formális nyelvek (pl. a programozási nyelvek) elemzése volt az első olyan feladat, amelyre attribútum nyelvtanokat alkalmaztak, azonban számos további, *valamilyen szabályszerűségeket mutató szekvenciális jelsorozat* alkalmas arra, hogy formális eszközök segítségével tanulmányozzák. Általában ilyenek például az élő szervezet működése közben keletkezett elektromos hullámok [KókI96], [KókP96], [Kók97].

A természetes nyelvi mondatok szerkezetének leírása (a természetes nyelvi szintaxis) az egyik legnagyobb kihívás. Számos akadálya van azonban annak, hogy egy adott természetes nyelvet csupán formális nyelvi átírási szabályokkal adjanak meg. Ezek közül néhány: a szükséges szabályok nagy száma, a számtalan kifejezési forma (nyelvjárás, szleng, szóhasználat), a természetes nyelvek szavainak többértelműsége stb. Ennek ellenére a természetes nyelvek feldolgozása során vannak olyan részfeladatok, ahol a szerző és munkatársai sikeresen alkalmaztak attribútum nyelvtanokat.

Az attribútum nyelvtanok hatékonyan felhasználhatók a számítógépes tanulás során is, például az AGLEARN tanuló rendszerben (lásd 4.6. Fejezet), ahol a magyar nyelv szófaji egyértelműsítésére készített alkalmazásban a szavak mondatbeli környezetét leíró attribútumok kiszámítására használták, amely attribútumok azután jótékonyan befolyásolták a C 4.5 tanuló rendszerrel kapott egyértelműsítési szabályok pontosságát. A korlátozott természetes nyelvi környezet megteremtésére tett kísérlet: egy attribútum nyelvtanokon alapuló természetes nyelvi interfész található a következő 4.3. Fejezetben.

4.3. Természetes nyelvi interfészek

Az alábbiakban ismertetésre kerül a szerző és munkatársai által kifejlesztett attribútum nyelvtan alapú természetes nyelvi interfész prototípus rendszer. Az elkészített prototípus a THALES nevet kapta és síkgeometriai szerkesztések elvégzését tette lehetővé. A fejezet teljes szövege megjelent angol nyelven az [Ale90] publikációban.[32]

A THALES rendszer a következőképpen működik: a szerkesztések utasítás sorozatok, minden utasítás egy-egy angol nyelvű mondattal van megfogalmazva. A felhasználó által begépelt mondatot egy természetes nyelvi elemző olvassa be, majd elkészíti a mondat egy belső reprezentációját. Ezután egy végrehajtó modul értelmezi, majd a kívánt tevékenység eredményét megjeleníti grafikus képernyőn. Lehetőség van a szerkesztések vagy a szerkesztések egy részének mentésére, ami lehetővé teszi a későbbi felhasználást, illetve egyes szerkesztési lépések beillesztését más alkalmazásokba. A szerkesztéseket a rendszer közönséges szövegfájlokban tárolja, amelyeket egyszerű szövegszerkesztő programokkal lehet feldolgozni. A rendszer része egy kidolgozott példákat tartalmazó példatár, amely tanárok és diákok számára teszi könnyebbé a geometriai szerkesztések tanítását illetve tanulását. A képernyőn megjelenő objektumokat a szerkesztések közben mozgatni lehet a látványosabb megjelenítés érdekében. Az objektumok azonosításában ez nem okoz bizonytalanságot.

A THALES természetes nyelvi interfész három fő modulból áll:

- Lexikális-morfológiai feldolgozó modul
- Szintaktikus elemző
- Szemantikus kiértékelő és végrehajtó

A nyelvi feldolgozás egy attribútumos leíráson alapul. Ebből a leírásból a PROF-LP fordító program generátor rendszer [Gyim88] készít elemző programot és attribútum kiértékelőt. A lexikon véges számú szót tartalmaz, amelyek lefedik a szerkesztések résznyelvét. A generált elemző program egy lexikális és egy szintaktikus elemző komponenst tartalmaz. A lexikális elemző morfológiai elemzést is végez, így a lexikon mérete minimális lehet. A szintaktikus modul egy felülről-lefelé (top-down) elemzőből áll, a jól ismert LL(1) algoritmus szerint működik, és amelyet attribútumokkal bővítettek ki a szemantikus feldolgozáshoz. A szemantikus kiértékelés és végrehajtás egy meta-szintű objektum leíráson alapul, amely lehetőséget biztosít

---

[32] A kutatási-fejlesztési projektet egy a gyakorlatban is használható termék elkészülésének reményében anyagilag támogatta az Egyesült Államokban bejegyzett Cogito Ltd., Philadeplhia cég, amiért a cikk szerzői itt szeretnék köszönetüket kifejezni.

geometriai relációk megvalósítására a szerkesztések különböző lépései során [AleA90]. Az objektum leírás szintén attribútum nyelvtan alapú, azonban nem teljes elemző programot, hanem egy forrásnyelvi modult generál, amelyet a THALES rendszer forráskódjával együtt kell lefordítani.

4.3.1. A THALES természetes nyelvi interfész nyelvészeti tulajdonságai

A rendszer szintaktikus és szemantikus kiterjedését az alábbi példákon lehet szemléltetni:

(1) *Draw two parallel lines.*
(2) *Label by E a point on the upper line.*
(3) *From point E, draw a perpendicular line to the lower line.*
(4) *Label the point of intersection of the lines by F.*
(5) *Draw a circle c around the midpoint of the segment EF.*
(6) *Inscribe an isosceles triangle in circle c.*
(7) *Construct a heptagon PQRSTUV.*
(8) *Draw two circles with radius EF at a distance equal to the difference between the base of the triangle and side RS og the heptagon.*
(9) *Inscribe a quadrangle that is tangent to the circle on the left part of the screen.*

4.2. Ábra. Példák a THALES rendszer által megértett mondatokra

Szintaktikailag az utasítások három fő elemet tartalmaztak:[33]

- parancsot jelentő ige, az állítmány
- a mondat tárgya
- határozók (specifier)

A határozó lehet Pre-specifier (elő-határozó), illetve Post-specifier (utó-határozó) a mondatban elfoglalt aktuális helye alapján. Egy mondaton belül mind a kettő, az elő-határozó és az utó-határozó is előfordulhat, amelyek együtt egy komplex határozót alkothatnak. A parancsot jelentő ige az elvégeztetni kívánt tevékenységet azonosítja. A mondat tárgya egy tárgyesetben lévő névszói szerkezetnek felel meg. Rendszerint a tárgy a képernyőn létrehozni és megjeleníteni kívánt alakzatot jelenti. A tárgy tartalmaz minden olyan névelőt, melléknevet, számnevet, utótagot, amely előfordulhat egy tárgy részeként. Ennek megfelelően a „*the triangle*" vagy a „*the upper and lower points of intersection of the sides of the two smaller isosceles triangles*" kifejezés lehet egy mondat tárgya. A határozó egy feltétel (megszorítás, restrikció), amely vagy elöljárós szerkezettel vagy egy alárendelő mondattal van megfogalmazva. A határozó típusától függően a megszorítást vagy az igére vagy a tárgyra kell alkalmazni. Az alábbi (10) és (11) mondatokban a megszorítást az igére illetve a tárgyra kell alkalmazni.

(10) *By connecting points A and B, draw a line.*
(11) *Construct a triangle inside the circle.*

Ez a két határozó típus egymást átfedheti, mivel az igére vonatkozó feltételek a készítendő objektumra vonatkozóan is tartalmazhatnak feltételeket, illetve fordítva a tárgyra vonatkozó feltétel befolyásolhatja a végrehajtás módját.

A síkgeometriai szerkesztések résznyelve nagy szabadságot nyújt az elöljárós szerkezetek használatára. Az elöljárós szerkezetek különböző kombinációi a parancsot jelentő igével és a tárggyal együtt ugyanazt az eredményt adják a végrehajtás szintjén. Jelen esetben némi könnyebbséget jelent az elöljárós szerkezetek feloldásánál, hogy nincs alany, csupán a parancsokban megengedett szerkezetek használhatók. Ebben a helyzetben az elöljárós szerkezetek redukálhatók velük megegyező jelentésű „such that" (olyat, ami) relációkká. A korábban említett (10) és (11) mondatok így átalakított (10') és (11') verziói:

(10') *Draw a line such that connects points A and B.*
(11') *Construct a triangle such that is inside the circle.*

A tárgy szerkezete, hasonlóan a határozóéhoz, állhat több egymást követő tárgy felsorolásából, amelyek így egymást követve egy komplex tárgyat alkotnak. A komplex tárgy feldolgozása különbözik a határozók feldolgozásától. A különbség oka az, hogy a komplex határozó egy megszorítás lista, amelyet egy vagy több

[33] Mivel minden mondat egy parancs volt, amely a számítógépet utasította, ezért a mondatokban nem szerepelt alany. Ezt úgy is mondhatjuk, hogy az alany minden esetben a „te" vagyis a számítógép volt.

tárgyra kell alkalmazni. A komplex tárgy pedig objektumok egy listája, amelyet meg kell konstruálni. A THALES rendelkezik egy korlátozott anaphora feloldási képességgel: egy névmással történő hivatkozást meg tud feleltetni a legutóbbi lépésben konstruált objektumnak.

A természetes nyelvi mondatokban megfigyelhető egy többértelműség ami a kötőszavas felsorolások miatt fordul elő. Ezt a többértelműséget egy dialógussal tudja csak a rendszer *„feloldani"*, a felhasználónak segítséget kell adnia a mondat szándék szerinti értelmezéséhez. Meg kell jegyezni, hogy ez a jelenség eredendően benne van az angol nyelvben, és nincs is algoritmus a szándékolt jelentés meghatározására. Egy példa látható erre a (12) mondatban.

(12) *Draw a line above the triangle and quadrilateral ABCD.*

A rendszer általánosíthatóságát jelentősen befolyásolják a prototípus rendszer készítésének körülményei és az implementáció egyes részletei. Ezek bemutatására kerül sor a következő fejezetben.

4.3.2. Az implementáció egyes kérdései

A gyakorlati szempontok mellett a fejlesztési folyamatot még két további szempont befolyásolta [Mart83]:

a) hordozhatóság más alkalmazások irányába
b) más természetes nyelvekre történő alkalmazás (német, francia, spanyol, magyar)

Hogy ezeket a követelményeket ki lehessen elégíteni (amelyek több szempontból is ellent mondanak egymásnak) a természetes nyelvi interfészt annyira általánosra kellett készíteni, amennyire csak lehetett. A következőkben megvizsgáljuk a THALES moduljait és egyenként tekintjük át az általánosítás lehetőségeit.

A modulok közül az első a lexikális-morfológiai feldolgozó modul. Egy előre adott gépi szótárral dolgozik. A ragozott szóalakok elemzése viszonylag egyszerű az angol nyelv szegényes morfológiája miatt. A módszer könnyen átvihető más alkalmazásokra, így kielégíti fenti a) követelményt. Más természetes nyelvre elkészíteni egy morfológiai elemző programot, például a magyarra, sokkal nehezebb feladat.

Ami a szintaktikus elemző modult illeti, a THALES elemzőt a PROF-LP rendszer generálja. Az elemző egy környezet-független, lényegében LL(1) nyelvtanon alapul. Az esetleges LL(1) konfliktusok feloldására kézzel írt előretekintő eljárások készültek. Ezek az eljárások egyszerűek, mert az input a szintaktikus elemzés fázisában már tokenizálva van. Mivel a THALES rendszer csak parancsokat kezel, ezért többé-kevésbé egyszerűen átvihető más alkalmazásokra, ahol a program által elvégzendő feladatok leírhatók utasításokkal (CAD rendszerek, szövegszerkesztők stb.). A különböző természetes nyelvű felszólítások nagyfokú hasonlóságának köszönhetően a THALES rendszer szintaktikai modulja valószínűleg átvihető más doménre, illetve kisebb mértékben más természetes nyelvekre. Véleményünk szerint az egyik legérdekesebb (és legnehezebb) kérdés a tevékenységeket teljesen átfogó természetes nyelvi interfész készítése, azaz a szemantika implementálása. A következő fejezet a THALES rendszer tervezésekor alkalmazott elvek kerülnek bemutatásra. A szemantikus modul általánosításának kérdései a következő fejezet végén kerülnek kifejtésre.

4.3.3. A THALES rendszer szemantikus kiértékelő és végrehajtó modulja

Első lépésként a tradicionális programozási nyelvekben ismert statikus (*static*, fordítási idejű) és a dinamikus (*dynamic*, futási idejű) szemantika közti különbségre hívjuk fel a figyelmet. A statikus szemantika az összes, fordításkor figyelembe vett környezet-függő tulajdonságok halmazaként definiálható. A legfontosabbak ezek közül:

- A változók definiáló és alkalmazott előfordulásainak megkülönböztetése
- Az azonosítási probléma: megtalálni az alkalmazott előforduláshoz tartozó definiáló előfordulását egy változónak
- A típus kompatibilitás kérdése

A dinamikus szemantikán azokat a tevékenységeket értjük, amelyeket a program futása során végrehajt. Fordító programok esetén van egy világos határvonal a statikus és a dinamikus szemantika között. Interpreterek esetében a két fajta szemantikát már nem olyan könnyű szétválasztani, bár egy jól strukturált interpreter belsejében a kettő jól elkülönül.

A továbbiakban megkíséreljük a THALES rendszer statikus szemantikáját definiálni. Ajánlásokat dolgozunk ki arra vonatkozóan hogy a statikus szemantika egy THALES-hez hasonló természetes nyelvi

interfészhez hogyan generálható egy magas szintű specifikáció alapján. Ami a dinamikus szemantikát illeti, az jelentős mértékben alkalmazásfüggő. Ezért a dinamikus szemantikát úgy tekintjük, hogy az kielégíti a fent említett b) pontot, azaz különböző természetes nyelvek között is hordozható.

A statikus szemantika a THALES esetén két részből áll. Az első rész felelős a környezet-független nyelvtannal megadott nyelvben szereplő attribútumok alapstruktúrájának kialakításáért. Ezt a feladatot kényelmesen el lehet végezni az attribútum nyelvtanos specifikáció alapján. Példaként itt látható egy kis részlet a THALES szintaktikus részéhez tartozó attribútum nyelvtanból.

```
Object = Np App Np_Rem ;
do
  Obj_Description <-- Create_Object(Obj_Description,
                                    Np.Np_Descripton,
                                    App.UnitLists,
                                    Np_Rem.OfLists) ;
end ;

Np_Rem = „of" Object ;
do
  OfLists := CreateOfList(Object.Obj_Description) ;
end ;

Np_Rem = ;
do
  OfLists := Nil ;
end ;
```

4.3. Ábra. A THALES attribútum nyelvtan egy részlete

A 4.3. ábra bemutatja a főnevek hierarchikus birtokos szerkezetét (amelyet az „*of*" elöljáró szó köt össze) leíró nemterminálisokat és a hozzájuk kapcsolódó attribútumokat. A `Create_Object` és a `Create_OfList` egy eljárás és egy függvény, amelyek egy lista szerkezetet alakítanak ki. Az `Obj_Description`, `Np_Description`, `UnitLists` és az `OfLists` szintetizált attribútumai az `Object`, `Np`, `App`, `Np_Rem` nemterminálisoknak ebben a sorrendben. Ezek az attribútumok írják le a szerkesztésekben szereplő objektumok tulajdonságait. Az `Np` (noun phrase) nemterminális a névszói szerkezetet jelöli, míg az `App` (apposition) az objektum opcionális azonosítóját.

A statikus szemantika másik része az objektumok definiálásáért és azonosításáért felelős. Ez a tevékenység egy szimbólumtáblán alapul, amely folyamatosan módosul a szerkesztési parancsok feldolgozása (fordítása és végrehajtása) során. Ez a modul az előző tevékenység során összegyűjtött és rendszerezett attribútum struktúrán alapul. Vizsgáljuk meg a statikus szemantikához tartozó tevékenységeket egy kis szerkesztési példán:

(13.1) *Draw an acute triangle and two points A and B.*
(13.2) *Draw an altitude of the triangle.*
(13.3) *Around the midpoint of the altitude, construct a circle inside the triangle.*

A következő kérdésekre kell válasz kapnunk:

1/a. *triangle* – ez egy definiáló vagy egy alkalmazott előfordulás-e?
1/b. lehet-e egy háromszög *acute*?
1/c. points – ez egy definiáló vagy egy alkalmazott előfordulás-e?
1/d. az *A* és *B* azonosítók száma egyezik-e a tárgyak számával?
2/a. a *triangle* objektumnak lehet-e *altitude* nevű komponense?
2/b. a *triangle* – ez egy definiáló vagy egy alkalmazott előfordulás-e?
ha alkalmazott előfordulásról van szó, akkor megkeresni a szimbólumtáblában.
2/c. *altitude* (magasság) – ez egy definiáló vagy egy alkalmazott előfordulás-e?
3/a. az *altitude* objektumnak lehet-e *midpoint* (középpont) nevű komponense?
3/b. *midpoint*, *altitude*, *circle*, *triangle* – ezek definiáló vagy alkalmazott előfordulások-e?
3/c. az *inside* reláció alkalmazható-e egy körre és egy háromszögre?

Nyilvánvalóan van kapcsolat a tradicionális programozási nyelvekben alkalmazott statikus szemantika és aközött, ahogyan a THALES természetes nyelvi interfész kezeli a szemantikus kérdéseket. A problémákat egy szimbólumtábla segítségével lehet megoldani. A szimbólumtábla elemei előre definiált típusú objektumok. A résznyelv főnevei megfeleltethetők a szimbólumtáblában használható objektumok típusainak. Formálisan egy *t* típust a következőképpen lehet leírni:

```
t :    Nouns
N1,1 ... N1,i1 : t1
.
.
.
Nn,1 ... Nn,in : tn

       Adjectives
A1, ... Ak

       Measure
M1, ..., Mt
```

(a) A fenti definícióban az $N_{i,j} = (k_{i,j}, s_{i,j}, f_{i,j})$, amelyben $f_{i,j}$ a *t* típus t_i résztípusának (komponensének) a neve, $s_{i,j}$ értéke 0, ha valódi komponensről van szó és az értéke 1, amennyiben egy a valódi komponensek adataiból származtatható másodlagos attribútumról van szó, a $k_{i,j}$ értéke a komponens példányainak száma egy objektumon belül: -1, ha nincs korlát a komponensekre (pl. a körben a sugarak száma), pozitív szám esetén a komponensek száma pontosan.

(b) t_i a komponens típus neve

(c) Az $A_i = (a_{i,1}, \ldots a_{i,j_i})$. Az a_{i,j_i} a *t* típusú objektum mellett álló lehetséges melléknevek listája. Egyszerre egy *t* típusú objektum mellett egy A_i melléknévcsoportból csak egyetlen $a_{i,j}$ állhat. A melléknevek sorrendje a rendszerben tetszőleges.

(d) Az $M_i = (m_1, \ldots m_{i,t_i})$. Az m_{i,t_i} mértékegységet jelöl, amely egy *t* típusú objektum mellett állhat. Egyszerre egy M_i mértékegység csoportból csak egyetlen $m_{i,t}$ állhat.

Az alábbiakban a háromszög (*triangle*) típus definíciójának egy részlete következik:

```
triangle : nouns
(3,0, vertex) (1,1, centroid) (3,1, midpoint)  : dot
(3,1, altitude) (3,1, edge) (3,1, center_line) : line

adjectives
(acute, right, obtuse)
(equiangular, isosceles, scalene)
...
```

Ha elkészítik a definíciókat a fentiek alapján, akkor a típus kompatibilitás, a melléknevek kezelése, a mértékegységek kezelése a specifikáció szintjén kezelhető. Továbbra is fennmarad azonban az objektumok közötti relációk kezelésének kérdése. Ezeket a relációkat a korábban említett határozók tartalmazzák. A határozók transzformálhatók egy *„olyat, ami"* relációvá. Egy formális definíciója lehetne a lehetséges relációknak a következő:

```
p1 ... pi : r { (t1,1, t1,k) (t2,1 ... t2,k) ... (tn,1 ... tn,k) }
```

például:

```
in, inside : in_something { (point, triangle) (point, circle)
                            (triangle, circle), (circle, triangle) ... }
```

Az objektumok definiáló és alkalmazott előfordulásai közötti különbségtétel a tárgy megadásakor adott névelőkből és appozíciókból (azonosítók) nyert információ alapján történik. Alkalmazott előfordulás esetén meg kell találni a szimbólumtáblában a megfelelő definiáló előfordulást. A tradicionális programozási nyelvekben az azonosítás egy relatíve egyszerű algoritmuson alapszik (vö. blokk struktúrájú nyelvek). A THALES-szerű természetes nyelvi interfészek esetében az alkalmazott előfordulás környezete hordozza azt az információt, ami alapján az azonosítás módszere kiválasztható. Az alábbiakban a THALES-ben alkalmazott főbb azonosítási eljárások találhatók:

Amennyiben az alkalmazott előfordulásnál meg van adva:

a) azonosító
A megadott azonosítóval rendelkező objektum megkeresése a szimbólum táblában.
(pl. *Draw a line from P.*)

b) objektum + azonosító
Csak a megadott típusú objektumok között keresi meg az adott azonosítóval rendelkezőt.
(pl. *Draw the radius of the circle c.*)

c) objektum
Visszafelé haladva az első előfordulását keresi meg a adott típusú objektumnak.
(pl. *Draw an altitude of the triangle.*)

d) melléknév lista + Objektum
Csak az adott típusú objektumok között keres, a melléknév listában felsorolt minden melléknévnek egyeznie kell a szimbólumtáblában szereplő objektum mellékneveivel.
(pl. *Draw an altitude of the acute isosceles triangle.*)

A fentiek során láttuk, hogy a statikus szemantika két részre osztható: a nyelvi struktúrák alap attribútumainak kiszámítására valamint a szimbólumtábla kezelésének technológiájára.

A nyelvi struktúrák alap attribútumainak kiszámítása:

- Kevéssé függ az alkalmazástól.
- Jelentős mértékben nyelv-függő (a speciális szintaxis miatt), ahol azonban maguk az attribútumok nagyjából azonosak.

A szimbólumtábla kezelés:

- Kevéssé függ az alkalmazástól. Egy új alkalmazásban csak a típusok és a relációk definícióit kell újraírni.
- Nagymértékben általánosítható, mivel csak a nyelvfüggő részét kell a típusok és a relációk definícióinak újraírni (a típusok nevét, a mellékneveket, stb.)

A dinamikus szemantika:

- Nagy mértékben függ az alkalmazástól.
- Az adott természetes nyelvtől teljesen független.

4.3.4. További lehetséges kutatási feladatok

A természetes nyelvi interfész prototípus fejlesztése közben szerzett tapasztalatok azt mutatták, hogy a teljes interfész specifikációja számára jobb lenne, ha azt egyetlen metanyelvi leírással lehetne leírni nem kettővel, mint ahogy az jelenleg történik. Ehhez azonban szükség lenne a PROF-LP fordítóprogram generáló rendszer [Gyim88] továbbfejlesztésére úgy, hogy az objektum-orientált specifikációkat is támogasson [Hed89], [Kos87].

Másfelől, a magas szintű programozási eszközök alkalmazhatóságának vizsgálata figyelmet kell fordítson más létező rendszerek természetes nyelvi képességeinek, kapacitásának a vizsgálatára és jobb megértésére. Ennek érdekében a szerzők vizsgálatokat folytatnak a PROF-LP és a Prolog összehasonlítására. Két Prolog rendszert vizsgáltak meg az Arity Prolog 5.1-et és az MProlog 2.3-at.[34]

4.4. Fonéma felismerés az IMPUT algoritmussal

Az alábbiakban ismertetésre kerülnek a szerző és munkatársainak tudományos eredményei a magyar magánhangzók felismerésével kapcsolatosan. A felismerési szabályok tanulására az IMPUT (lásd 3. Fejezet) abduktív tanuló algoritmust használták fel. A fejezet eredetileg angol nyelven jelent meg az [Ale97] publikációban.

Az elkészített kísérleti rendszerben a spektogram modellt használták, mivel az könnyen visszaalakítható beszéddé. A szerzők megvizsgálták, hogy a spektogram grafikus reprezentációjának megváltoztatása miként befolyásolja beszédjelet. Ennek a munkának a fő célja az volt, hogy meghatározzák a spektogram legfontosabb részeit. A feldolgozás következő lépése néhány akusztikus paraméter (pl. pozíciók, intenzitások, lokális maximumok alakja) meghatározása a spektogramból egy genetikus algoritmussal [Jel95]. Az akusztikus

[34] A cikk elkészültekor ezek voltak a beszerezhető Prolog rendszerek.

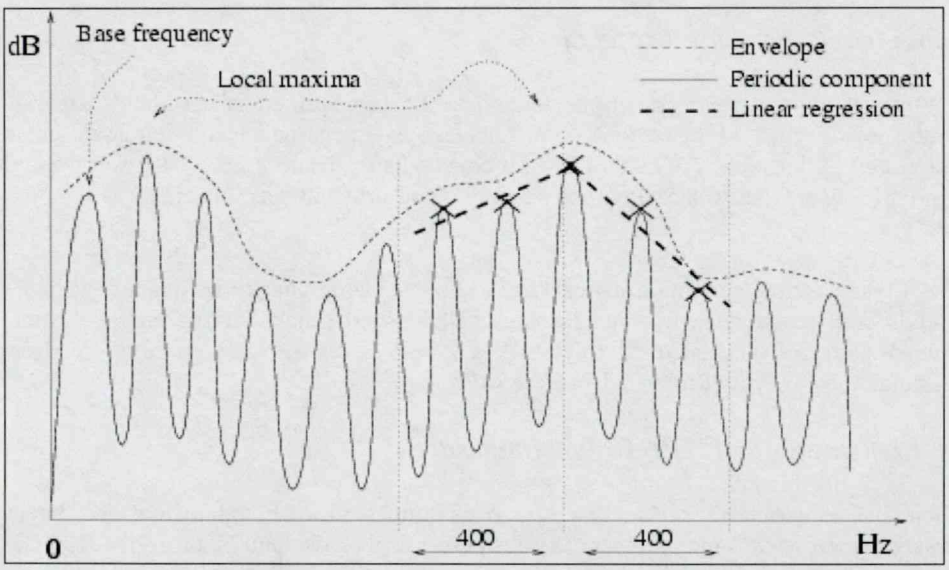
paraméterekből egy szabályrendszert határoztak meg a különálló magyar magánhangzók felismerésére. A szabályokat Prolog klózokkal reprezentálták és az IMPUT [AleI97] tanuló algoritmussal finomították. A kapott eredményeket összehasonlították az általánosan elfogadott magyar fonetikai kézikönyv [Bol95] adataival. A megfigyelések szerint az egy adott beszélőre optimalizált szabályok jobb eredményt adtak, mint az általános szabályok. Az egy beszélőre adaptált rendszerek előnye természetesen ismert a szakirodalomban [Rab93]. Az itt ismertetésre kerülő megközelítés fő előnye, hogy egy viszonylag egyszerű tanulási modellt sikerült felállítani magyar magánhangzók felismerési szabályainak tanulására. Nagy valószínűséggel a modellt más fonémákra is lehet majd alkalmazni. A hosszú távú célja ezeknek az erőfeszítéseknek azonban egy teljes, folyamatos beszédet felismerni képes rendszer elkészítése magyar nyelvre.

4.4.1. Az akusztikus paraméterek meghatározása

A fonetikában a magánhangzók jellemzésének tradicionális módja a *formánsaik* leírása. Ebben a munkában, azonban ettől egy kissé megközelítést használtak az alább ismertetendő okok miatt. Jól ismert tény, hogy egy (statikus, önmagában kiejtett) magánhangzó egy kvázi-periodikus jel, amelyet egy speciális burkológörbe határol [Rab78]. A periodikus komponens frekvenciája felelős a magánhangzó hangszínéért és a burkológörbe hordozza azt az információt, ami lehetővé teszi a magánhangzó felismerését. Általában elfogadott [Rab78], hogy – legalábbis a magánhangzók esetében – a burkológörbén található lokális maximumok helye és nagysága elegendő az osztályozás elvégzéséhez. Az itt felsorolt fogalmak magyarázata a 4.4. ábrán látható. A formánsok első definícióját ezek után megfogalmazhatjuk.

4.4. Definíció: Formáns: a lokális maximumokhoz tartozó *frekvenciák* értéke. A lokális maximumokban felvett intenzitás a *formáns intenzitása*. A szokások jelölés a következő: az F0 jelöli az alapfrekvenciát, az F_i pedig az i^{th} maximum helyet növekvő sorrendben.

A kiszámíthatóság szempontjából a fenti definíció nem teljesen elegendő, ugyanis néhány magánhangzó esetében, különösen az [u] és az [o] magánhangzóknál, bizonyos *formánsok* (rendszerint az F1 és F2) annyira közel esnek egymáshoz, hogy egyetlen lokális maximumot hoznak létre a burkológörbén. Emlékezzünk rá, hogy a burkológörbe csak a periodikus jel maximumhelyein ismert (lásd 4.4. ábra).

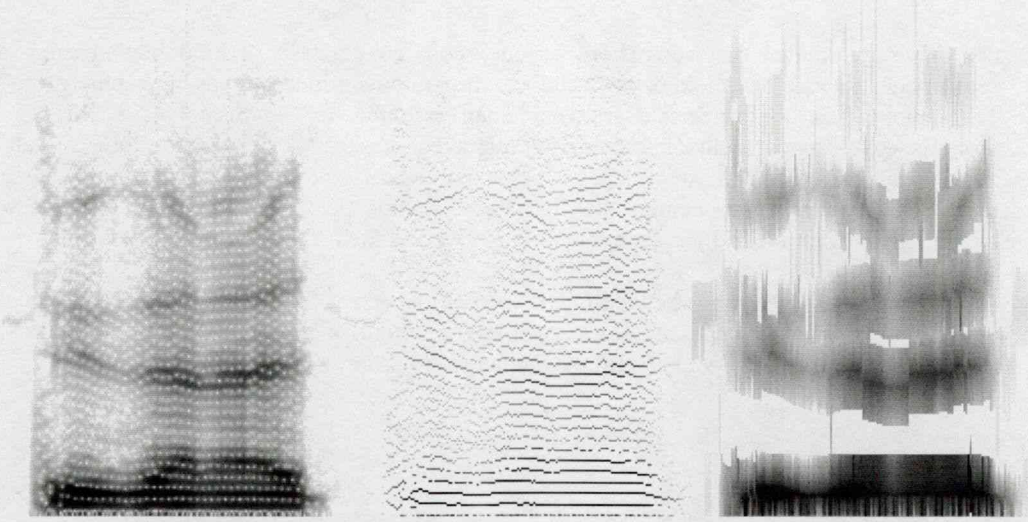


4.4. Ábra. Néhány alapfogalom, és a meredekség paraméter meghatározása

A fentiek miatt nem a formánsok, hanem a lokális maximumhelyek kerültek felhasználásra. A lokális maximumhelyekhez az alábbi négy tulajdonság tartozik: *hely, intenzitás, bal oldali meredekség, jobb oldali meredekség*. A hely és az intenzitás szokásos paraméterek, míg a bal és a jobb oldali meredekség a csúcs alakját hívatott jellemezni.

Minden tulajdonságot a magánhangzó mintákhoz tartozó eredeti beszéd jelek spektrális felbontásából számítottak ki. Nyilvánvalóan először a burkológörbét kell meghatározni. Erre a célra egy genetikus algoritmus alapú módszert használtak a GAS [Jel95] egy adaptációját. A GAS egy függvény-struktúra felderítő algoritmus különböző heurisztikákkal bővítve, amely képes egy adott pontonként megadott függvény többszörös lokális

maximumhelyeit is megtalálni. Jelen esetben a megadott függvény a beszédjel spektrális felbontása volt (mint egy a frekvenciától függő leképezés). A megtalált maximum helyek a burkológörbe egy approximációját adják ezen a módon. A folyamat a 4.5. ábrán látható.



4.5. Ábra. Az akusztikus jelfeldolgozás lépései. Az első kép a [tizɜnɜj] **magyar szó (tizenegy) eredeti spektogramját, a második a periodikus komponens maximumhelyeit (a burkológörbe közelítését), a harmadik kép maximumhelyek négy adatából (hely, intenzitás, bal és jobb oldali meredekség) generált spektogramot ábrázolja minden egyes időszeletre.**

A GAS program a maximumhelyek jellemzéséhez szükséges négy paraméter közül az első kettőt meg tudja határozni: a *lokális maximumhelyet* és az *intenzitást*. Ebben a lépésben egy további heurisztikát is használtak: amennyiben két lokális maximumhely túl közel van egymáshoz, kevesebb mint 400 Hz a két frekvencia különbsége, akkor az alacsonyabb frekvenciájút törölték. A heurisztika alkalmazhatóságát pszichofizikai kutatások alátámasztják [Chis79].

Végül a bal és jobb oldali meredekséget lineáris regresszióval határozták meg, a burkológörbe azon pontjainak felhasználásával, amelyek 400 Hz-nél kisebb távolságban voltak a lokális maximumhelytől (lásd 4.3. ábra és 4.4. ábra).

Megjegyezzük, hogy a fenti paraméterek mindegyike a beszédjel egy időszeletére vonatkozik. Mivel a tanulás egy magában álló kitartott magánhangzóra vonatkozik, amely sok egymás után álló időszeletből áll, így a paraméterek globális statisztikai tulajdonságai használhatók. Miután a paramétereket minden időszeletre kiszámították, nyilvánvalóan a lokális maximumhelyek csoportosítása (*clustering*) következik. Egy következő időszelet lokális maximumhelye akkor kerül be egy csoportba *(cluster-be)*, ha a csoportbeli elemek átlaga és az új elem távolsága kevesebb mint 200 Hz. Ellenkező esetben egy új egy elemű csoportot fog alkotni a szóban forgó új maximumhely.

A végső eredmény így egy 9 adatot tartalmazó lista (a maximumhelyek csoportjainak listája): a már ismert négy adat átlaga és szórása minden egyes csoportra, valamint egy százalékos arány amely azt mutatja meg, hogy az összes időszelet hány százalékát reprezentálja ez a csoport.

4.4.2. Felismerési szabályok tanulása a magyar magánhangzók felismerésére

A tanulási feladat során a magyar magánhangzók felismerésének problémáját kívánták megoldani. A tizennégy magyar magánhangzó közül ötöt választottak ki az [u], [o], [ɔ], [a], és az [ɜ]. Minden egyes minta ugyanattól a férfi beszélőtől származott. Minden egyes magánhangzóra több példafájl állt rendelkezésre, amelyek különbözhettek az intonációban valamint a hangmagasságban. Bár a minták statikus magánhangzók voltak (mintegy 1 másodpercesek), a fenti különbségek minden egyes időszeletre is érvényesek, mivel a beszélő nem tud hosszabb ideig azonos spektrumú magánhangzókat kiejteni.

A célkitűzés az volt, hogy minden egyes magánhangzóra egy-egy felismerő szabályhalmazt találjanak Prolog programok formájában. A fonetikai kézikönyvben [Bol95] található szabályokat próbálták ki először, de azok hatékonysága gyengének bizonyult a mintákon. Ennek az lehet a fő oka, hogy azok a szabályok inkább

elméletiek voltak, és ezért nem voltak képesek alkalmazkodni a speciális mintákhoz. Ezután a kézikönyvben alkalmazott szabályokat általánosították és kaptak egy túl-általános (overly-general) programot minden egyes magánhangzóra. Majd erre a kezdeti programra alkalmazták az IMPUT [AleI97] algoritmust, amely finomította a szabályrendszert.

A kezdeti program minden magánhangzóra hasonló volt, amit azután öt különböző tréning adatsorral tanítottak, végül pedig kaptak öt különböző specializált programot, minden egyes magánhangzóra egyet. A kiindulási programok csak a formánsok frekvenciáiban különböztek egymástól. Az IMPUT rendszer alkalmazásakor, az egyes magánhangzókhoz pozitív mintaként használták fel a beszélőtől felvett az adott magánhangzóra vonatkozó mintákat, míg negatív példaként használták fel az összes más magánhangzóra vonatkozó mintát. Így minden egyes tanulási konfiguráció 5 pozitív példából és 20 negatív példából állt. Az eredményeket egy ugyanekkora teszt halmazon ellenőrizték. Az eredmények értékelését lásd később.

A továbbiakban az egyszerűség kedvéért *formánsnak* nevezzük a burkológörbe tapasztalati lokális maximumhelyeit. A formánsokat a spektogram alapján egy genetikus algoritmus határozta meg (lásd a 4.4.1. Fejezet). Az alábbiakban látható egy példa az [ɔ] magánhangzó formánsaira:

```
Field              Value          Variance

Size of cluster : 95.169082%
Center (Hz):     131.603923      45.998686
Left angle :       1.488087       0.019766
Right angle :     -1.288563       0.013596
Intensity :      246.050761       5.114175

Size of cluster : 96.135266%
Center (Hz):    2363.782683     439.968458
Left angle :       1.388186       0.000902
Right angle :     -1.226952       0.021302
Intensity :      121.030151      58.642307

Size of cluster : 83.091787%
Center (Hz):    3556.233019   2642.078360
Left angle :       1.345919       0.010351
Right angle :     -1.125229       0.028088
Intensity :      135.505814     114.575548
```

4.6. Ábra. Néhány alapfogalom, és a meredekség paraméter meghatározása

A fenti ábrán öt egymás utáni sor határoz meg egy formánst. A `Size` paraméter a formáns hossza a teljes beszédjel hosszához viszonyítva (hány időszeleten át figyelhető meg), a `Center` paraméter a formáns középfrekvenciája, a `Left angle` és a `Right angle` a csúcs bal és jobb oldali meredeksége, az `Intensity` pedig a formáns intenzitása egy logaritmikus skálán (decibelben).

A `Variance` (szórás) értékek a tanulási folyamatban nem vesznek részt, kivéve a középfrekvencia esetében. A tanulási folyamat számára a lebegőpontos értékeket diszkrét értékekké konvertálták, vagy Prolog atomokra, vagy pedig egész számokra (a szögek és a frekvencia esetén). Minden egyes formánst egy hat tagból álló struktúrával reprezentáltak. A komponensek a következők voltak: a diszkrét méret, egész frekvencia, a frekvencia varianciájának diszkretizált értéke, a bal és a jobb oldali meredekség egész fokokban, diszkrét intenzitás.

Egy magánhangzó minta formánsok egy halmazából áll, ezért egy mintafájlhoz a fenti hat adattal jellemzett formánsok listája rendelhető. A lista egy eleme egy formánsnak felel meg. Például az [ɔ] magánhangzó lehetséges formánsai a következő Prolog termmel írhatók le:

```
sound( [(fulllength,  125, veryveryshort, 85, 110, loud),
        (fulllength, 2350, veryshort, 75, 110, noise),
        (fulllength, 3550, wide, 75, 120, noise)] ).
```

Amint az már említésre került, a [Bol95] kézikönyvből származó általános felismerési szabályokat általánosítani kellett a magyar magánhangzók esetében, mert nem viselkedtek kielégítően a példákon. Ezt a következőképpen végezték el: eredetileg a szabályok három szomszédos formáns (az F1, F2, F3) egyidejű

jelenlétét követelték meg. A kezdeti programban egy új F4 formánst vezettek be, illetve a négy formáns közül csak kettő egyidejű jelenlétét követelték meg. A formánsok középfrekvenciája nem változott. Az [ɔ] magánhangzóhoz készített kezdeti program a következőképpen néz ki:

```
a_vowel(L) :- a_exist_formant_1(L), a_exist_formant_2(L).
a_vowel(L) :- a_exist_formant_1(L), a_exist_formant_3(L).
a_vowel(L) :- a_exist_formant_1(L), a_exist_formant_4(L).
a_vowel(L) :- a_exist_formant_2(L), a_exist_formant_3(L).
a_vowel(L) :- a_exist_formant_2(L), a_exist_formant_4(L).
a_vowel(L) :- a_exist_formant_3(L), a_exist_formant_4(L).

a_exist_formant_1(List) :-
            member((S, Hz, HzW, _, _, Int), List),
            a_accept_size_1(S), a_accept_hertz_1(Hz),
            a_accept_hzwidth_1(HzW), a_accept_int_1(Int).

a_exist_formant_2(List) :-
            member((S, Hz, HzW, _, _, Int), List),
            a_accept_size_2(S), a_accept_hertz_2(Hz),
            a_accept_hzwidth_2(HzW), a_accept_int_2(Int).
...
a_accept_size_1(Size) :- size(Size).
a_accept_hertz_1(Hertz) :- a_formant_1(Hertz).
a_accept_hzwidth_1(HertzW) :- width(HertzW).
a_accept_int_1(Int) :- intensity(Int).

a_accept_size_2(Size) :- size(Size).
a_accept_hertz_2(Hertz) :- a_formant_2(Hertz).
a_accept_hzwidth_2(HertzW) :- width(HertzW).
a_accept_int_2(Int) :- intensity(Int).
...
a_formant_1(Hertz) :- between(Hertz, 500, 750).
a_formant_2(Hertz) :- between(Hertz, 900, 1250).
...
size(fulllength).
size(long).
size(short).
size(veryshort).
size(veryveryshort).
...
```

4.7. Ábra. Az IMPUT tanuló algoritmus számára készített kezdeti hipotézis vázlata

A 4.7. ábrán látható kezdeti hipotézis lefedte az összes pozitív és negatív példát is. Ezért a hipotézis specializációjára van szükség, amelyet az IMPUT tanuló algoritmussal oldottak meg. Az IMPUT [AleI97] egy abduktív algoritmus, amely az unfolding transzformációt és a klóz törlést használja fel a kezdeti hipotézis módosítására. A tanuló algoritmus tartalmaz egy beépített hibakereső alrendszert [Kók94], [Paa94], amely megkönnyíti a szükséges transzformáció helyének megtalálását. Az IMPUT rendszer működéséhez szükség van egy *bölcsre* is, aminek köszönhetően a tanuló algoritmus hatékonyabb eredményt nyújt (kisebb és pontosabb programot kaphatunk).

Az eredmények értékeléséhez szükséges leszögezni, hogy nem minden magánhangzót lehet megkülönböztetni. Egyes magánhangzók azonosíthatók [ɔ] vagy [a] magánhangzóként is, mások pedig [o] vagy [u] magánhangzóként. A döntés sok esetben szubjektív.

A példa kedvéért nézzük az [ɔ] magánhangzót. A kezdeti hipotézis lefedte mind az 5 pozitív és 15-öt a negatív példák közül. Az első unfolding transzformáció után kaptunk egy olyan hipotézist, amely már csak 2 negatív példát fedett le, mindkettő az [a] magánhangzóhoz tartozó minták közül került ki. Az input példákat jobban megvizsgálva az látható, hogy az [ɔ] és az [a] magánhangzókhoz tartozó minták kevéssé különböznek egymástól. Ebben az esetben nem lehetett további specializációt találni, amelynek segítségével egy *konzisztens hipotézis* (2.7. definíció) keletkezett volna.

Amikor az eredményül kapott programokat a teszt példákon futtatták, akkor az ott megadott pozitív példákat mind felismerték az eredményül kapott programok, azonban a negatív példák közül 4-5-öt is lefedtek.

Nagyjából minden magánhangzóra hasonló eredmények adódtak, kivéve az [ɔ] magánhangzót, ahol az eredmény rosszabb volt. Bár a hibásan felismert magánhangzók száma jelentős mértékben csökkent a kezdeti hipotézishez viszonyítva, az eredményül kapott hipotézissel sem lehetünk elégedettek. Egy megoldás lehet a specializáció folytatása további tanulási példák alkalmazásával. Bár, egy jó felismerő rendszernek kevés számú mintából is tudnia kell beszélő-specifikus szabályokat meghatározni. Ezért a későbbiekben a kezdeti felismerő program bővítésével próbálkoznak a szerzők, új típusú akusztikus paraméterekkel és újabb szabályokkal.

4.4.3. Hasonló rendszerek és további kutatási feladatok

A szabályalapú és a sztochasztikus rendszerek azóta versenyeznek egymással, amióta az első beszédfelismerő rendszereket tanulmányozták [Wai90], [Zue85]. Az általánosan elfogadott kritika a tudás-alapú rendszerekkel szemben az, hogy nem eléggé flexibilisek. Sok szerző ezt a hátrányt úgy próbálja meg kiküszöbölni, hogy *fuzzy* döntési logikát vagy neurális hálózatokat épít be a rendszerébe [Hueb94]. A nyolcvanas években a szakértői rendszer nagyon népszerű volt [Mam95] de mivel a több forrásból származó információ egyesítését sosem sikerül teljesen megérteni, ezek a rendszerek sem hoztak igazi javulást. Mindezen hátrányok ellenére, sok tudásalapú beszédfelismerő rendszerről publikáltak a legmodernebb minta-illesztéses (*pattern-matching*) módszereket felülmúló eredményeket. Néhányban ezek közül [Hueb94]

A bemutatott módszert a magánhangzó felismerés megoldásához vezető hosszú út első állomásának kell tekinteni. A cél az IMPUT ILP tanuló rendszer kipróbálása volt, hogy alkalmazható-e egy olyan nehéz és bonyolult kérdés megoldására, mint a beszédfelismerés. Meg kell említeni, hogy a beszédfelismerésen belül a magánhangzó felismerés az a terület, ahol a klasszikus minta-illesztés a legjobb eredményt adja. Igazán drámai fordulatot itt nem lehetett várni. A tudás-alapú rendszerek ott múlják felül a sztochasztikus rendszereket, ahol finom fonetikai megkülönböztetés szükséges, mint például a stop mássalhangzók felismerése [Lam93]. A következő feladat a rendszer további bővítése, a megfelelő akusztikus jellemzők meghatározása minden lehetséges fonéma típushoz. Menet közben azokat az eljárásokat is ki kell fejleszteni, amelyekkel az új jellemzők kinyerhetők a beszédjelből. Ezután kerülhet sor olyan szabályok megalkotására, mint amilyenek ebben a fejezetben is előfordultak, amelyek leírják az akusztikus jellemzők észlelését és a konverziójukat fonémákra. Ezután az egész rendszert egy általános beszédfelismerő keretrendszerbe kell illeszteni. Az ILP módszerek számos esetben alkalmazhatók lesznek ezeknek a szabályoknak a finomítására.

4.5. A magyar természetes nyelvi szófaji egyértelműsítése

A következő 4.6. Fejezetben és a 4.7. Fejezetben a gépi tanuló algoritmusok egy speciális alkalmazásáról esik szó. Mind a két fejezet a magyar nyelv szófaji egyértelműsítésének problémájával foglalkozik. Ebben a fejezetben a tanuló algoritmusok alkalmazásához szükséges tréning adatbázis fontosabb tulajdonságai kerülnek bemutatásra.

A szófaji egyértelműsítés[35] a természetes nyelv feldolgozás egyik fontos első lépése. A feldolgozást végző számítógépes rendszer beolvassa egy mondat szavait majd a szavakhoz hozzárendeli a morfo-szintaktikai címkéjüket.[36] Ezt a tevékenységet *szófaji annotációnak*, angolul *tagging*-nek nevezik és a mondatelemzésben kulcsszerepet játszik. Mivel minden természetes nyelvben vannak többértelmű szavak, amelyeknek több lehetséges (leggyakrabban 2–4) címkéje lehet, ezért minden szófaji annotáló szoftver kell tartalmazzon egy egyértelműsítő modult, a morfológiai analizátor modul mellett. Az egyértelműsítő modul feladata a szövegkörnyezetnek megfelelő címke kiválasztása a lehetséges címkék közül.

Több lehetséges megközelítés ismert az egyértelműsítést végző modulra nézve: a két leggyakoribb a valószínűségi (HMM, Hidden Markov Model), illetve a szabályalapú. A dolgozatban olyan szabályalapú megközelítésről esik majd szó, amelyben a szabályokat egy gépi tanuló algoritmusok állítja elő. A szerző és munkatársai több tanuló algoritmust vizsgáltak meg, tanulási modelleket készítettek, a tanuló algoritmusok alkalmazásához szükséges szoftver keretrendszert elkészítették. A dolgozatban ismertetett eredményeket egy kézzel egyértelműsített magyar korpusz, a TELRI [TELRI] korpusz feldolgozásával kapták. A továbbiakban korpusz fontosabb tulajdonságai következnek.

---

[35] Angolul Part-Of-Speech tagging, illetve röviden POS-tagging

[36] Pl. *csináltam*: főige, cselekvő, múlt idő, egyes szám első személy, alanyi vagy tárgyas ragozás.

4.5.1. A tréning adatbázis

A dolgozat későbbi fejezeteinek elkészítéséhez a szerző és munkatársai a magyar TELRI korpusz [TELRI] többszörösen javított változatát használták. A korpusz első változatát a "MULTEXT-East" (MULtilingual TEXT tools and Corpora for Eastern and Central European Languages) COPERNICUS projekt keretében készítették 1995–1997 között. A korpuszban a további javításokat a készítők: az MTA Nyelvtudományi Intézet munkatársai végezték. A "MULTEXT-East" projekt célja az volt, hogy bemutassa az eredetileg angolszász nyelvekre kidolgozott morfológiai kódrendszer, az MSD (Morpho-Syntactic Description) alkalmazhatóságát a kelet európai természetes nyelvekre is. A projektben egyetlen magyar partner volt: az MTA Nyelvtudományi Intézete. A projektben az MSD kódrendszert kibővítették hat kelet- és közép-európai nyelv (bolgár, cseh, szlovén, román, észt, magyar) jellegzetességeivel. *Ezzel egy nagymértékben nyelv-független és az európai nyelvek többségében használható morfológiai jelölésrendszert hoztak létre.*

A projekt keretében létrehozott korpusz alapja Orwell ismert regénye az *1984* volt, amelyet számos természetes nyelvre fordítottak le. A fenti COPERNICUS projekt során az *1984* c. regényt tokenekre bontották, szófaji elemzést végeztek a szavakon, majd elvégezték a szöveg egyértelműsítését. A szavak morfológiai kategorizálására az újonnan kidolgozott MSD kódokat használták fel [Erja97]. A 4.8. táblázatban a magyar ún. *Orwell korpusz* méret adatai láthatók.

Kategória	A tokenek száma		
	Tréning	Teszt	Együtt
Szó	59035	21673	80708
Írásjel	12484	5235	17719
	71519	26908	98427

4.8. Táblázat A tokenek száma a magyar *Orwell korpuszban*

A magyar nyelv az ún. agglutináló nyelvek közé tartozik, azaz nagyon sok jelet és ragot használ, emiatt a magyar szavak nagyon sok (néhány ezer) különböző MSD kódot kaphatnak. Például a főneveknek 1324 különböző MSD kódja lehet a lehetséges kapcsolódó jeleknek és ragoknak megfelelően; a mellékneveknek 2772, a névmásoknak is megközelítőleg 3000 kódja lehet. Ez azt jelenti, hogy például a főneveket 1324 részosztályba sorolja a kódrendszer.

Hogy ezt a számot csökkenteni lehessen, az MTA Nyelvtudományi Intézetében Oravecz és munkatársai egy redukált rendszert fejlesztettek ki, amely csak 125 különböző kategóriát tartalmaz [Ora98]. Nyilvánvalóan, a CTAG rendszer használata információvesztéssel jár, de mint azt Oravecz [Ora98] leírta, a CTAG rendszer alkalmazása mégis hasznos és több alkalmazása is ismert magyar szövegek annotálására. A TELRI korpusz, amelynek egy kis részlete látható a 4.9. ábrán tartalmazza mind a CTAG mind pedig az MSD annotációkat, ezért mind a két kódolással végezhetünk kísérleteket. A tanulási példák készítéséhez hol a CTAG kódokat, hol pedig az MSD kódokat használták.

A TELRI [TELRI] magyar korpusz egy SGML[37] fájl, amely a CES[38] DTD (Document Type Definition) séma alapján készült. A regényben 4 nagyobb fejezet van, amelyek közül az első kettő tréning, a második kettő pedig teszt adatbázisként szolgált. A korpusz fájl struktúrája követhető a 4.9. ábrán. A teljes regény bekezdésekre, mondatokra, azon belül szavakra van bontva. A bekezdéseket `<par>` és `</par>` szimbólumok (tagek), a mondatokat `<s>` és `</s>` szimbólumok, a tokeneket `<tok>` és `</tok>` szimbólumok határolják. A fájlban minden egyes fejezethez, részhez, bekezdéshez, mondathoz egy egyértelmű azonosító tartozik pl.

[37] SGML (Standard Generalized Markup Language), az ISO 8879 szabvány szerinti *szabványos általánosított markup nyelv* http://www.iso.ch. A *markup* kifejezés azt jelenti, hogy az adatfájlban metainformációt is lehet elhelyezni pl. az adatok szerkezetéről. Ennek a technikai megvalósítására szabadon definiálható, egymásba is ágyazható `<Meta>` (kezdő) és `</Meta>` (záró) szimbólumok ún. *tag-ek* szolgálnak. Egy SGML fájlhoz minden esetben hozzátartozik a lehetséges *tag* struktúrát definiáló DTD. Az SGML szabványból fejlődött ki az XML (Extensible Markup Language), amely egy számítástechnikai szempontból könnyebben kezelhető részhalmaza az SGML-nek.

[38] CES (Corpus Encoding Standard) http://www.cs.vassar.edu/CES/.

'Ohu1.1.2.1' (Orwell, Hungarian, 1. fejezet, 1. rész, 2. bekezdés, 1. mondat). A tokenek nincsenek azonosítóval ellátva.

```
<par from='Ohu.1.2.1'>
  <s from='Ohu.1.2.1.1'>
    <tok type=WORD>
      <orth>Der&uuml;lt</orth>
      <disamb><base>der&uuml;lt</base><msd>Afp-sn</msd><ctag>AN</ctag>
      </disamb>
      <lex><base>der&uuml;lt</base><msd>Afp-sn</msd></lex>
      <lex><base>der&uuml;l</base><msd>Vmis3s---n</msd></lex>
    </tok>
    <tok type=PUNCT>
      <orth>,</orth>
      <ctag>COMMA</ctag>
    </tok>
    <tok type=WORD>
      <orth>hideg</orth>
      <disamb><base>hideg</base><msd>Afp-sn</msd><ctag>AN</ctag></disamb>
      <lex><base>hideg</base><msd>Afp-sn</msd></lex>
    </tok>
  ...
  </s>
  <s from='Ohu.1.2.1.2'>
  ...
  </par>
  <par from='Ohu.1.2.2'>
  ...
```

4.9. Ábra A magyar korpusz fájl (SGML, CES DTD) egy részlete

A tanuló algoritmusok futtatásához a korpusz Prolog formáját használták. Minden mondatnak megfelel egy `s/2` term, amelyben az első argumentum helyén a mondat azonosító, a másodikon pedig a mondatbeli szavak listája található. Egyjelentésű szavak esetén a szó helyén az MSD vagy a CTAG kód áll, többjelentésű szavak esetén pedig egy *pár* áll. A párban a második elem a szó lehetséges címkéinek listája, az első helyen pedig mondat környezetnek megfelelő címke áll. A morfológiai elemzés után, az egyértelműsítés elvégzése előtt, nincs még kijelölve a helyes címke, ekkor a pár első helyén egy *undefined* szimbólum áll. Ezzel a megoldással egy Prologban könnyen kezelhető adatbázis formátumot kapható.

```
s('Ohu.1.2.1.1', [('Afp-sn', ['Afp-sn', 'Vmis3s---n']), 'WPUNCT', 'Afp-sn',
'Afp-sn', 'Nc-sn', ('Vmis3s---n', ['Afp-sn', 'Nc-sn', 'Vmis3s---n']), 'WPUNCT',
('Tf', ['P---sn', 'Tf']), 'Nc-pn', 'Rg', 'Mc-sa', ('Vmis3p---n', ['Afp-pn',
'Vmis3p---n']), 'SPUNCT']).
```

4.5.2. Az MSD morfológiai kódrendszer

Az MSD kódrendszer a szavak legkülönbözőbb attribútumainak kódolására szolgál, az európai nyelvek többségére alkalmazható. A szavak morfo-szintaktikai attribútumait (típus, nem, szám, személy, eset) egy karaktersorozat reprezentálja.

Kategória	Kód	Kategória	Kód
Melléknév	A	Particle	Q
Kötőszó	C	Határozószó	R
Determináns	D	Névutó	S
Indulatszó, hangutánzó szó	I	Névelő	T
Számnév	M	Ige	V
Főnév	N	Egyéb, ismeretlen szó	X
Névmás	P	Rövidítés	Y

4.10. Táblázat Az MSD kód első betűje, a szó fő kategóriája

A magyar *asztalnak* szó az `Nc-sg------` kódot kapja, amely a következőket jelenti: főnév, köznév, egyes szám, birtokos eset (*genitivus*). A hiányzó vagy az adott természetes nyelvben nem értelmezhető

attribútumok helyén - jel áll[39]. Az első pozíción a szó fő kategóriája áll, a lehetséges kódokat a 4.10. Táblázat tartalmazza. (Az MSD kódokról részleteket a [Erja97] irodalom tartalmaz) Eszerint a *hideg* szó kódja például `Afp-sn`.[40] Ez egy olyan melléknevet jelent, amelynek a következő attribútumai vannak: kvalifikatív, pozitív, egyes szám, alanyeset. A szavak számát az egyes főkategóriákban a 4.11. táblázat tartalmazza.

4.5.3. A CTAG kódrendszer

Kategória	A tokenek száma		
	Tréning adatok	Teszt adatok	Összesen
Melléknév (A)	7157	2382	9539
Kötőszó (C)	5408	1994	7402
Indulatszó (I)	72	17	89
Számnév (M)	1046	323	1369
Főnév (N)	14881	5170	20051
Névmás (P)	4552	1923	6475
Határozószó (R)	7869	2969	10838
Névutó (S)	827	280	1107
Névelő (T)	6856	2336	9192
Ige (V)	10288	4254	14542
Egyéb (X)	13	12	25
Rövidítés (Y)	66	13	79
	59035	21673	80708
Írásjelek	12484	5235	17719
	71519	26908	98427

4.11. Táblázat: A tokenek száma az *Orwell korpuszban* fő kategóriánként (szófajonként)

Azért, hogy az MSD-ben előforduló nagyon sok lehetséges kategória számát csökkenteni lehessen, Oravecz [Ora98] bevezette a CTAG kódokat, amelyben már csak 120 szó kategória van, 4 írásjel kategória és 1 kód van fenntartva az ismeretlen szavak számára. Az 4.12. táblázatban található a CTAG kódok listája.

```
CTAG kódok szavakra:
APA, APAX, APAY, APD, APDX, APDY, APN, APNX, APNY, APO, APOX, APOY, ASA,
ASAX, ASAY, ASD, ASDX, ASDY, ASN, ASNX, ASNY, ASO, ASOX, ASOY, CP, I, MD,
MP, MPX, MPY, MS, MSX, MSY, NPA, NPAX, NPAY, NPD, NPDX, NPDY, NPN, NPNX,
NPNY, NPO, NPOX, NPOY, NSA, NSAX, NSAY, NSD, NSDX, NSDY, NSN, NSNX, NSNY,
NSO, NSOX, NSOY, PPA, PPAX, PPAY, PPD, PPDX, PPDY, PPN, PPNX, PPNY, PPO,
PPOX, PPOY, PSA, PSAX, PSAY, PSD, PSDX, PSDY, PSN, PSNX, PSNY, PSO, PSOX,
PSOY, RG, RO, RP, RQ, RV, ST, T, VA, VMCP1P, VMCP1S, VMCP2, VMCP2P, VMCP2S,
VMCP3P, VMCP3S, VMIP1P, VMIP1S, VMIP2, VMIP2P, VMIP2S, VMIP3P, VMIP3S,
VMIS1P, VMIS1S, VMIS2, VMIS2P, VMIS2S, VMIS3P, VMIS3S, VMMP1P, VMMP1S,
VMMP2, VMMP2P, VMMP2S, VMMP3P, VMMP3S, VMN, X, Y.

Írásjel CTAG kategóriák: OPUNCT, CPUNCT, WPUNCT, SPUNCT.

Az ismeretlen szó CTAG kategóriája: UNKNOWN.
```

4.12. Táblázat A magyar szavak CTAG kategóriái

A kódszó első betűje itt is a fő kategóriát jelenti, mint az MSD-ben, míg a fennmaradó hosszú kódsorozatot sokkal rövidebbekkel helyettesítette. Például az `NPNX` jelentése *főnév*, amelynek a következő attribútumai vannak: többes szám, alanyeset, valamilyen birtokos személyrag vagy jel. A második betű `P` vagy `S` kódolja a többes illetve az egyes számot. A harmadik betű `N`, `A`, `D` vagy `O` kódolja az eseteket, nevezetesen az alanyesetet, tárgyesetet, részeshatározós esetet, illetve az egyéb esetet. A negyedik betű a birtokos végződés meglétét jelzi, de

[39] Például a magyar nyelvben nincs a névszóknak *neme*, ellenben a szláv nyelvekben van.

[40] A kódszó végén lévő - jeleket nem kell kiírni, ezért az `Afp-sn-------` kódot rövidíthetjük `Afp-sn`-nel.

nem kódolja a birtokos személyét, számát, a tulajdonolt tárgy számát. Az Orwell korpuszban található főnév kódok eloszlását a 4.13. táblázat mutatja be.

4.5.4. A tréning adatbázis kialakítása

Az eredeti magyar korpuszt Prolog formára konvertálták. Ehhez először egy olyan lexikont készítettek, amely tartalmazott minden a szövegben szereplő szóalakot és a morfo-szintaktikai kódjaikat. Ha egy szó több morfo-szintaktikai kóddal is rendelkezhetett, akkor azt annyiszor szerepeltették a lexikonban. Egy második lépésben az SGML fájlt Prolog formájúra alakították a lexikon felhasználásával. A konverzió mondatonként történt, amelyeket az SGML fájlban a <s> és </s> tag-ek alapján azonosítottak.

A főnév részosztályok száma az Orwell korpuszban			
CTAG kód	Tréning	Teszt	Összesen
NPA	304	106	410
NPAX	106	30	136
NPAY	0	0	0
NPD	50	9	59
NPDX	37	7	44
NPDY	0	0	0
NPN	832	279	1111
NPNX	193	70	263
NPNY	4	1	5
NPO	470	128	598
NPOX	119	36	155
NPOY	1	3	4
NSA	1310	475	1785
NSAX	555	206	761
NSAY	4	0	4
NSD	359	90	449
NSDX	131	35	166
NSDY	0	0	0
NSN	5286	1928	7214
NSNX	1319	445	1764
NSNY	13	10	23
NSO	2686	932	3618
NSOX	1098	380	1478
NSOY	4	0	4
Összesen	14881	5170	20051

4.13. Táblázat Az Orwell korpuszban a főnév részosztályok száma

Például, az Orwell regény első mondata ez volt:

Derült, hideg áprilisi nap volt, az órák éppen tizenhármat ütöttek.

Ha a CTAG kódolást használjuk, akkor a mondat szavaihoz az alábbi morfológiai kategóriákat rendelhetjük (a Prolog programozási nyelvben való könnyebb kezelhetőség miatt a kódokat kisbetűs változataikra konvertálták):

```
s('Ohu.1.2.1.1', [(asn, [asn, vmis3s]), wpunct, asn, asn, nsn, (vmis3s, [asn, nsn,
vmis3s]), wpunct, (t, [psn, t]), npn, rg, ms, (vmis3p, [apn, vmis3p]), spunct]).
```

Ha a tekintett szó egyértelmű, akkor egyszerűen helyettesítették a CTAG kódjával (pl. *hideg* – `asn`, *áprilisi* – `asn`, a vessző írásjel kódja – `wpunct`, idő – `nsn`). Amennyiben a tekintett szó nem volt egyértelmű, akkor egy párt feleltettek meg neki, amelynek az első eleme a helyes tag, a második elem pedig a lehetséges tag-ek listája volt (pl. *derült* – `(asn, [asn, vmis3s])`, vagy *volt* – `(vmis3s, [asn, nsn, vmis3s])`).

Ugyanez MSD kódokkal a következő:

```
s('Ohu.1.2.1.1', [('Afp-sn', ['Afp-sn', 'Vmis3s---n']), 'WPUNCT', 'Afp-sn', 'Afp-
sn', 'Nc-sn', ('Vmis3s---n', ['Afp-sn', 'Nc-sn', 'Vmis3s---n']), 'WPUNCT', ('Tf',
['Pd--sn', 'Tf']), 'Nc-pn', 'Rg', 'Mc-sa', ('Vmis3p---n', ['Afp-pn', 'Vmis3p---
n']), 'SPUNCT']).
```

Ezen a módon mind a trénig (a regény 1. és 2. Fejezete), mind pedig a teszt (a regény 3. és 4. Fejezete) adatokat átalakították. A regényben szereplő 6768 mondatból 4583 tartozott a tréning és 2185 mondat a teszt adatokhoz. A különböző tanuló algoritmusokhoz (AGLEARN, C 4.5, Progol) szükséges adatfájlokat alapvetően a Prolog formájú korpuszból állították elő SICStus Prolog[41] programozási nyelven írt programok segítségével. Az elemzett korpusz tüzetesebb vizsgálatával megállapítható, hogy a szövegben fellelhető bizonytalanságokat mely szavak okozzák.

4.5. Definíció: Bizonytalansági osztály: egy bizonytalansági osztályba tartoznak mindazon tokenek, amelyeknek ugyanazok a lehetséges morfo-szintaktikai címkéik. Egy bizonytalansági osztályba rendszerint több hasonló jelentésű szó tartozik: pl. az `[asn, ms]` osztályba tartozik a: *temérdek*, *számtalan*, *harmincas*, *negyvenes*, ..., az `[asn, nsn]` osztályba tartozik az: *álló*, *dongó*, *metsző*, *vezető*, stb.

A bizonytalansági osztályok száma az Orwell korpuszban					
Osztály	Előfordulások száma		Osztály	Előfordulások száma	
	Tréning	Teszt		Tréning	Teszt
[asn, ms]	55	12	[ms, rg]	70	19
[asn, nsn]	96	27	[ms, t]	751	222
[asn, nsn, psn]	52	22	[nsn, psn]	111	52
[asn, nsn, vmis3s]	50	30	[nsn, rg]	59	18
[asn, psn]	68	33	[nsn, rg, rp]	112	46
[asn, rg]	92	23	[nsn, vmip3s]	52	41
[asn, vmis3s]	490	182	[psn, pso]	69	38
[aso, rg]	74	32	[psn, rp]	143	57
[cp, pso]	50	30	[psn, t]	1867	620
[cp, pso, rg]	87	44	[pso, rg]	217	85
[cp, rg]	880	294	[pso, rg, rp]	93	45
[cp, rg, vmip3s]	247	125	[rg, rp]	150	59
[cp, rp]	334	149	[rg, st]	285	100
[cp, vmis3s]	113	38			
			Összesen	6667	2443
		A korpuszban összesen volt		7229	2677

4.14. Táblázat. A leggyakoribb bizonytalansági osztályok száma (amelyek legalább 50-szer előfordultak a tréning adatok között)

A szerző saját eredményei közé tartozik a bizonytalansági osztályok bevezetése és alkalmazása a tanulási modellben. *Figyeljük meg, hogy a különböző bizonytalansági osztályokra megfogalmazott tanulási feladatok egymástól függetlenek!* Amennyiben a tanulási feladatokat bizonytalansági osztályonként készítik el, az jelentősen megnöveli a figyelembe vehető tanulási adatbázis lehetséges méretét. Egy tanulási feladatban felhasználható tréning adatok számát leginkább a számítógépek gyorsasága határozza meg. A tanuló algoritmusok műveletigénye legtöbbször nem lineárisan függ a példák számától, ezért a tréning példák számának csökkentése jelentősen rövidítheti a végrehajtási időt. A feladat felosztásával és több számítógép bevonásával még hamarabb kapható meg a végeredmény.

A bizonytalansági osztályok között vannak nagymértékben elterjedtek. A 4.14. táblázat, azt mutatja be, hogy CTAG kódolás esetén az Orwell korpuszban milyen a leggyakoribb bizonytalansági osztályok eloszlása. A táblázat alapján látható, hogy a felsorolt 27 osztály tartalmazza az előforduló bizonytalanságok több mint 90%-át (6667-et a 7229-ből: 92,22%, illetve 2443-at a 2677-ből: 91,60%). Ez azt is jelenti, hogy a legelterjedtebb

[41] Pontosabban SICStus Prolog 3.7.1. verziójával, ami a cikkek írásakor a legújabb volt, a Prolog rendszer Tcl/Tk grafikus interfésze is nagy szerepet kapott a programok felhasználói interfészének programozásában.

osztályokra adott pontos egyértelműsítési szabályok segítségével már egy jó hatásfokú egyértelműsítő program készíthető.

A bizonytalansági osztályok száma az Orwell korpuszban					
	Előfordulások száma			Előfordulások száma	
Osztály	Tréning	Teszt	Osztály	Tréning	Teszt
[Afp-pn, Nc-pn]	55	19	[Nc-sa, Nc-sn]	177	50
[Afp-pn, Vmip2p---n, Vmis3p---n]	51	25	[Nc-sa, Rg]	65	44
[Afp-pn, Vmis3p---n]	290	103	[Nc-sa---s3, Nc-sa-----s, Nc-sn]	193	77
[Afp-s2, Nc-s2, Rg]	52	21	[Nc-sn, Nc-s2---s3, Nc-s2-----s]	61	23
[Afp-sn, Afp-sp, Afp-sw]	70	23	[Nc-sn, P---sn]	158	70
[Afp-sn, Ccsw, Vmis3s---n]	113	38	[Nc-sn, Rg]	960	417
[Afp-sn, Mc-sn]	69	13	[Nc-sn, Rp]	119	46
[Afp-sn, Nc-sn]	860	233	[Nc-sn, Vmip3s---n]	197	76
[Afp-sn, Nc-sn, P---sn]	53	23	[Nc-sn, Vmis3s---y]	129	68
[Afp-sn, Nc-sn, Vmis3s---n]	1014	315	[Nc-sp---s3, Nc-sp-----s]	66	20
[Afp-sn, P---sn]	98	43	[Nc-sw, Rg]	65	37
[Afp-sn, Rg]	58	27	[P---s2, Rg]	58	26
[Afp-sn, Rg, Vmip3s---n]	55	10	[P---s9, Rg]	161	52
[Afp-sn, Vmip3s---n, Vmis3s---n]	140	39	[P---sm, Rg]	107	46
[Afp-sn, Vmis3s---n]	2659	941	[P---sn, P---sp]	282	148
[Afp-sn----s1, Vmis1s---n, Vmis1s---y]	50	8	[P---sn, Rp]	143	57
[Afp-sp, Afp-sw]	264	85	[P---sn, Tf]	1867	620
[Afp-sp, Afp-sw, Rg]	78	28	[P---ss, Rg, Rp]	93	44
[Afp-sp, Rg]	128	30	[Rg, Rp]	88	24
[Afp-sw, Rg]	53	13	[Rg, St]	139	55
[Ccsw, I]	297	115	[Vmcp3s---n, Vmip3s---n]	243	70
[Ccsw, P---sm, Rg]	87	44	[Vmip2p---n, Vmis3p---n]	63	23
[Ccsw, Rg]	880	295	[Vmip3p---y, Vmip3s---n]	120	78
[Ccsw, Rp]	334	149	[Vmip3p---y, Vmmp3p---y]	71	37
[Ccsw, Vmip3s---n]	247	125	[Vmip3s---y, Vmmp3s---y]	150	58
[Ccsw, Y]	827	269	[Vmis1s---n, Vmis1s---y]	50	24
[Mc-sn, Ti]	751	222	Összesen	15458	5546
		A korpuszban összesen volt		18784	6742

4.15. Táblázat. A leggyakoribb bizonytalansági osztályok száma (amelyek legalább 50-szer előfordultak a tréning adatok között)

A 4.14. táblázat MSD verziója a 4.15. táblázat. Ebben ismét azok az osztályok szerepelnek, amelyek legalább 50-szer előfordultak a tréning adatok között. A CTAG kódokhoz képest a bizonytalanságok abszolút száma is növekedett, több mint a duplájára. Az osztályok száma ugyancsak növekedett a korábbi 27 helyett 54-re. Az 54 osztály már csak mintegy 80%-át reprezentálja az összes bizonytalanságnak (15458-at a 18784-ből: 82,29%, illetve 5546-ot a 6742-ből: 82,26%). Néhány osztály megmaradt változatlanul pl. `[ms, t]` és az `[Mc-sn, Ti]`, vagy a `[psn, t]` és a `[P---sn, Tf]` osztályok, mások sok kis osztályra hasadtak szét. Az `[ms, t]` osztályban csak egy szó van az *egy* (számnév illetve határozatlan névelő). A `[psn, t]` osztályban ugyancsak egy szó van az *az* (a mutató névmás és a határozott névelő).

4.6. Szófaji egyértelműsítés az AGLEARN módszerrel

Ebben a fejezetben az AGLEARN tanuló algoritmus kerül bemutatásra, valamint az algoritmus egy alkalmazása a magyar természetes nyelv szófaji egyértelműsítésére. A fejezet anyaga angol nyelven megjelent az [AleW99] publikációban.

Az AGLEARN tanuló algoritmus feladata, hogy szemantikus egyenleteket tanuljon egy attribútum nyelvtan szabályaihoz. A tanulási folyamatban háttér szemantikus egyenletek és példák felhasználására van lehetőség. Az AGLEARN rendszer alábbiakban ismertetett alkalmazásában a feladat a szavak helyes szófaji címkéjének kiválasztása volt. A tréning korpusz[42] mintegy százezer szó méretű, kézzel egyértelműsített szöveg volt, amelyet tanításra és tesztelésre is használtak. Az AGLEARN algoritmus a C 4.5 [Quin93] attribútum érték tanuló program számára állított elő tanulási feladatokat. A generált adatok a mondatok struktúrájával kapcsolatban álló információkat tartalmaztak. Egy háttér attribútum nyelvtannal határozták meg ezeket az új attribútumokat. A tapasztalatok szerint a kibővített adatokból a C 4.5 tanuló rendszer pontosabb egyértelműsítési szabályokat tanult.

Az attribútum nyelvtanok a környezet-független nyelvtanok egy kiterjesztésének tekinthetők, amelyekben a nyelvtani szimbólumokhoz (a terminálisokhoz és a nemterminálisokhoz) attribútumokat rendelhetünk, a nyelvtani szabályokhoz pedig szemantikus egyenleteket, amelyek az attribútumok értékét definiálják. Több különböző attribútum kiértékelési módszert írtak már le [Alb91], [Der88], azonban kevés figyelem fordult magának az attribútum nyelvtannak a hatékony előállítására. Az attribútum nyelvtan definiálása általában sok emberi erőfeszítést igényel, ezért nagy segítséget jelentene egy olyan segédprogram, amely az attribútum nyelvtan szemantikus egyenleteit tanulási példák segítségével meg tudná határozni. Az attribútum nyelvtan szintaxisának definíciója azonban továbbra is emberi munka marad. A bemutatásra kerülő AGLEARN algoritmus [Gyim97] ezt a feladatot oldja meg. A szemantikus egyenletek definíciója tartalmazhat relációkat, ezért ez a feladat nem könnyű. Ezt a megközelítést a logikai programok és az attribútum nyelvtanok között meglévő szoros kapcsolat motiválta [Der85], [Der93], [Paa90]. A kapcsolatot a nemterminálisok és predikátumok közti megfeleltetés adja, valamint egy jól ismert formalizmus a logikai programozásban, amely hasonlít az attribútum nyelvtan jelölésrendszeréhez, nevezetesen a DCG (Definite Clause Grammars) [Per80]. Az AGLEARN módszer koncepciója hasonlít az ILP (Inductive Logic Programming) keretében kifejlesztett tanulási módszerekhez [Dzer93], [MuggI], [Mugg94]. Az ILP-ben a logikai programozási környezet nyújtja az eszközt a tanulási példák, a háttértudás, illetve a megtanulandó fogalmak definíciójához. A logikai programozási nyelv sokkal kifejezőbb, mint a propozíciós nyelvek, amelyeket egyes ILP rendszerekben használnak. Az ILP rendszerekben hatékonyabban lehet a háttértudást felhasználni, a háttértudás segítségével bonyolultabb objektumokat és fogalmakat felépíteni. Az AGLEARN eljárásban is ugyanez a volt módszer, de a reprezentáció különböző: a háttértudást és a fogalmakat egy attribútum nyelvtan írja le. A fejlesztő által kidolgozott attribútum nyelvtan szintaxis és az adott háttér tudás együtt lecsökkentheti a keresési teret és gyorsabb, valamint egyszerűbb reprezentációt adhat. Ebben az új megközelítésben a háttértudást nyelvi megszorításként használják (lásd 2.3.1. Fejezet). Az AGLEARN esetében egy példa tartalmaz egy mondatot, amely levezethető egy vizsgált nemterminálisból, továbbá a vizsgált nemterminális attribútumai is ki vannak számítva a példában. Azt is feltételezzük, hogy a szükséges környezet-független nyelvtan is adott. Ezek után az AGLEARN feladata, hogy az átírási szabályhoz tartozó szemantikus egyenleteket meghatározza. A tanulás során a környezet-független nyelvtant, háttér szemantikus egyenleteket és a példákat lehet felhasználni. Az itt ismertetésre kerülő megközelítésben S-attribútumos és L-attribútumos attribútum nyelvtanok használhatók és egyszerű szemantikus egyenletek tanulhatók. Megjegyezzük azonban, hogy a háttértudás ennél lényegesen komplexebb attribútum nyelvtant is tartalmazhat például OAG [Kas80].

Az alábbiakban ismertetésre kerülő alkalmazásban az AGLEARN tanuló algoritmust a magyar szövegek szófaji egyértelműsítésére alkalmazták. Az egyértelműsítés nem triviális feladat, mivel sok magyar szónak több különböző jelentése is van.[43] Az AGLEARN algoritmus feladata egy olyan szabályrendszer meghatározása volt, amely a szavakhoz a mondatban a szövegkörnyezetnek megfelelő szófaji címkét rendeli. A továbbiakban röviden áttekintjük magát az AGLEARN algoritmust, majd az egyértelműsítési feladat megoldását. Az AGLEARN részletesebb, minden részletre kiterjedő ismertetése megtalálható a [Gyim97] irodalomban.

4.6.1. Fogalmak és definíciók

Az attribútum nyelvtan fogalmát Knuth vezette be 1968-ban (lásd 4.1. definíció). Ugyancsak Knuth-tól

---

[42] A számítógépes természetes nyelv feldolgozásban korpusznak nevezik az annotált szövegadatbázisokat, az olyan számítógépes állományokat, amelyekben a szöveg strukturálisan tagolt (fejezetekre, bekezdésekre, mondatokra és szavakra bontott), az egyes szövegelemekhez egyértelmű azonosítót, valamint a további feldolgozás céljából címkéket (annotációkat) illesztettek.

[43] Pl. a *múlt* szó lehet ige, főnév vagy melléknév különböző mondatokban.

származik a csak szintetizált attribútumokat megengedő, ún. S-attribútumos nyelvtanok definíciója (lásd 4.2. definíció). Mivel az S-attribútumos nyelvtanok csak szűk körben alkalmazhatók, ezért a gyakorlatban szükség volt egy nagyobb osztályra: az L-attribútumos nyelvtanok osztályára (lásd 4.3. definíció). Az alábbiakban bemutatásra kerül egy attribútum nyelvtan, amely az egyszerű aritmetikai kifejezések típusát képes kiszámítani.

V_N = { Expression, Term, Factor, AddOp, MulOp }
V_T = { Real, Integer, +, -, ×, /, (,), }
A start szimbólum S = Expression

A szemantikus domén $\mathcal{T}$ = { T_{mode}, $T_{operator}$ }, ahol
T_{mode} = { int, real }
$T_{operator}$ = { add, sub, mul, div }

A függvény típusok: $\mathcal{F}$ = { add, sub, mul, div, int, real, id, f_1, f_2 }, ahol az add, sub, mul, div, int, real: konstansok
id: $T_{mode} \rightarrow T_{mode}$ az azonosság-függvény

f_1: $T_{mode} \times T_{mode} \rightarrow T_{mode}$, ahol
$f_1(op_1, op_2)$:= **if** op_1 == *real* **or** op_2 == *real* **then** *real* **else** *int*

f_2: $T_{operator} \times T_{mode} \times T_{mode} \rightarrow T_{mode}$, ahol
$f_2(op_1, op_2, op_3)$:= **if** op_1 == *mul* **and** op_2 == *int* **and** op_3 == *int* **then** *int* **else** *real*

Az attribútumok:
A ={ mode, operator }, csak szintetizált attribútumok vannak, $AI(V)=\emptyset$, minden $V \in V_N$.
AS(Expression) = AS(Term) = AS(Factor) = { mode },
AS(AddOp) = AS(MulOp) = { operator }.

Az átírási szabályok és a szemantikus egyenletek:

1: $Expression_0 \rightarrow Expression_1$ AddOp Term
$R(1)$: $Expression_0$.mode := f_1($Expression_1$.mode, Term.mode)

2: Expression → Term
$R(2)$: Expression.mode := id(Term.mode)

3: $Term_0 \rightarrow Term_1$ MulOp Factor
$R(3)$: $Term_0$.mode := f_2(MulOp.operator, $Term_1$.mode, Factor.mode)

4: Term → Factor
$R(4)$: Term.mode := id(Factor.mode)

5: Factor → Real
$R(5)$: Factor.mode := real

6: Factor → Integer
$R(6)$: Factor.mode := int

7: Factor → "(" Expression ")"
$R(7)$: Factor.mode := id(Expression.mode)

8: AddOp → "+"
$R(8)$: AddOp.operator := add

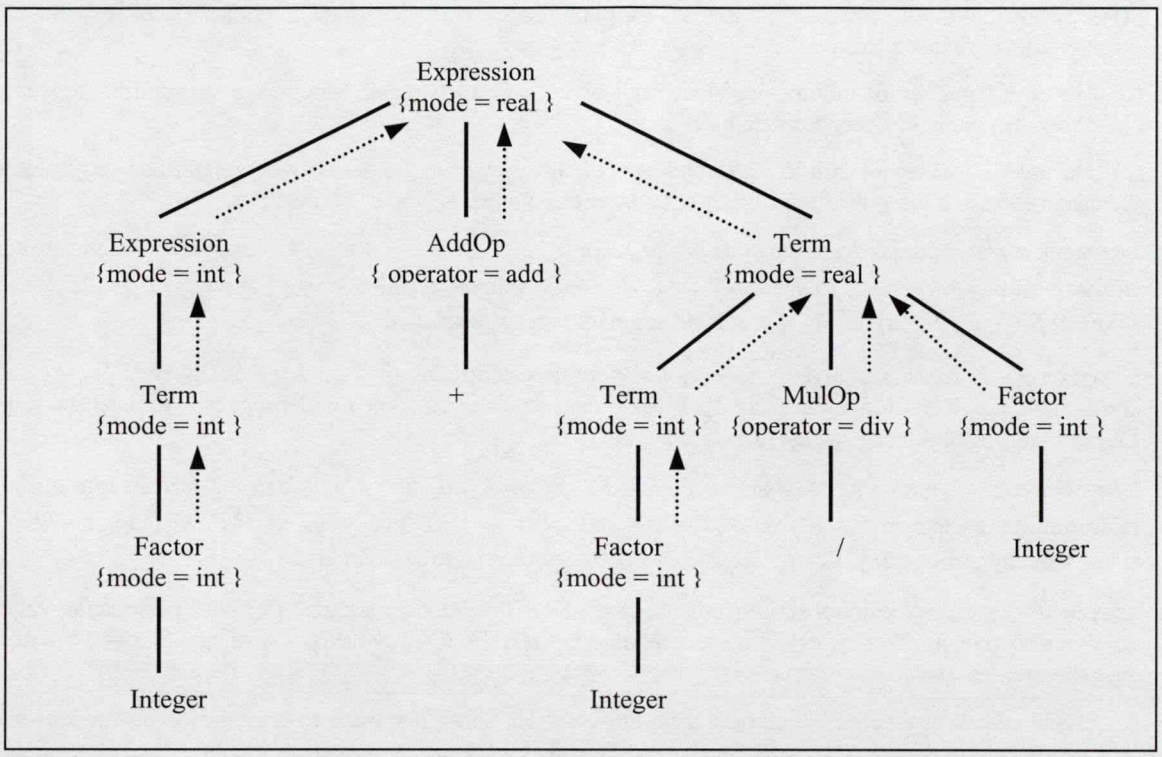
9: AddOp → "-"
$R(9)$: AddOp.operator := sub

10: MulOp → "*"
$R(10)$: MulOp.operator := mul

11: MulOp → "/"
$R(11)$: MulOp.operator := div

4.16. Ábra. Az egyszerű aritmetikai kifejezés típusát kiszámító attrbútum nyelvtan

A 4.16. ábrán látható attribútum nyelvtant felhasználva kapható a 4.17. ábrán bemutatott attribútumokkal dekorált levezetési fa. Az `Integer+Integer/Integer` inputra kapjuk az ábrán látható fát. A pontozott nyilak mutatják az egyes attribútum példányok közötti függéseket, továbbá láthatók az ábrán az attribútum példányok értékei is. Egy attribútum példány értéke akkor számítható ki, ha az összes olyan attribútum példány értéke rendelkezésre áll, amelytől az függ. Ahhoz, hogy bármely levezetési fában az attribútum példányok értékei kiszámíthatók legyenek, szükséges egy kiértékelési stratégia felállítása. Egyszerű attribútum nyelvtanok esetében, mint az S-attribútumos, vagy az L-attribútumos nyelvtanok, ez a stratégia automatikus. A példában szereplő attribútum nyelvtan S-attribútumos, mivel csak szintetizált attribútumokat használ.



4.17. ábra az `Integer+Integer/Integer` kifejezés attribútumokkal dekorált levezetési fája

4.6.2. Szemantikus függvények tanulása, egy S-attribútumos nyelvtan esetén

Az attribútum nyelvtanok szemantikus függvényeinek (és szemantikus feltételeinek) tanulásával foglakozik az alábbi fejezet. Az bemutatásra kerülő módszer ötletét a LINUS [Dzer93] ILP tanuló algoritmus adta. Az AGLEARN működése az alábbi három lépésből áll:

- A szemantikus függvények tanulásának problémájának átalakítása propozícionális tanulási problémává
- Egy propozícionális tanuló algoritmus alkalmazása a feladat megoldására
- A megtanult propozícionális hipotézis visszaalakítása szemantikus függvénnyé (szemantikus feltétellé)

A tanuló algoritmus feladata egy $p \in P$ átírási szabály bal oldalán álló nemterminális szintetizált attribútumaihoz kapcsolódó szemantikus egyenlet meghatározása. Legyen ez az átírásai szabály a következő: p: $X_{p0} \rightarrow X_{p1}\ X_{p2}\ \ldots\ X_{pn_p}$ és tegyük fel, hogy a feladat az X_{p0} nemterminális egy a szintetizált attribútumának a megtanulása. Jelölje továbbá $E_p^+(a)$ és $E_p^-(a)$ az $X_{p0}.a$ attribútumhoz tartozó pozitív és negatív példák halmazát úgy, hogy amennyiben $(w, (a, v)) \in E_p(a)$, akkor $X_{p0} \Rightarrow X_{p1}\ X_{p2}\ \ldots\ X_{pn_p} \Rightarrow^* w$, az attribútum értéke pedig v. Ahhoz, hogy az $X_{p0}.a$ attribútumhoz tartozó szemantikus egyenlet tanuló algoritmussal meghatározható legyen, egy $T(a)$ táblázatot kell konstruálni, a táblázat minden sora egy példának felel meg az egyik $E_p(a)$ halmazból. A táblázatnak a következő oszlopai vannak:

$$\{class, word, target\} \cup \mathcal{U} \cup \mathcal{F}_{UR} \cup \mathcal{F}_{UCR} \cup \mathcal{F}_{UF} \cup \mathcal{F}_{UCF}$$

amennyiben az adott sorhoz tartozó példa $e \in E_p(a)$, $e=(w, (a, v))$, az oszlopok a következőképpen definiáltak:

1. $class(e) = +$, ha $e \in E_p^+(a)$, $class(e) = -$, ha $e \in E_p^-(a)$.
2. $word(e) = w$
3. $target(e) = v$, a példában szereplő attribútum érték.
4. Legyen a $p \in P$ átírási szabályban $X_{pk}.b$ egy attribútum előfordulás $(0 < k \quad n_p)$, akkor van egy $X_{pk}.b$ attribútumnak megfelelő oszlop az $\mathcal{U}$-ban. Az oszlopban szereplő értéket a háttér attribútum nyelvtanban szereplő szemantikus egyenletekkel lehet kiszámítani. A jelen esetben az $e = (w, (a, v))$ példával ez a számítás a következő módon megy végbe:
   (a) Egy attribútumos elemzési fát épít a w input jelsorozat alapján, a G nyelvtan és az R szemantikus egyenletek felhasználásával.
   (b) Ha az X_{pk} gyökérből induló részfában csak olyan csúcsok vannak, amelyek a szemantikus egyenletek definiálva vannak, akkor a részfa kiértékelhető.
   (c) Ha az X_{pk} gyökérből induló részfában van olyan csúcs, amelyhez tartozó szemantikus egyenlet nem ismert, akkor a hiányzó attribútumot a rendszer egy *bölcstől* (oracle) kérdezi meg.
5. Legyenek a $p \in P$ átírási szabályban az $X_{pk_1}.a_1, X_{pk_2}.a_2, \ldots, X_{pk_l}.a_l$ $(1 \quad k_1, \ldots, k_l \quad n_p)$ alkalmazott attribútum előfordulások és legyen $f \in FUNC\text{-}SET, f: \tau_1 \times \ldots \times \tau_l \rightarrow \{true, false\}$ boolean függvény, $\tau_i = TYPE(X_{pk_i}.a_i)$, akkor az $f(X_{pk_1}.a_1, X_{pk_2}.a_2, \ldots, X_{pk_l}.a_l)$, reláció számára van egy oszlop az $\mathcal{F}_{UR}$-ben.
6. Legyen a $p \in P$ átírási szabályban $X_{pk}.b$ egy attribútum előfordulás $(0 < k \quad n_p)$, és legyenek $\{c_1, c_2, \ldots, c_m\}$ konstansok, amelyek előfordulnak az $X_{pk}.b$ oszlopban az $\mathcal{U}$-ban. Akkor minden egyes c_i konstansra van egy $\{X_{pk}.b == c_i\}$ reláció az $\mathcal{F}_{UCR}$-ben $(1 \quad i \quad m)$.
7. Legyenek a $p \in P$ átírási szabályban az $X_{pk_1}.a_1, X_{pk_2}.a_2, \ldots, X_{pk_l}.a_l$ $(1 \quad k_1, \ldots, k_l \quad n_p)$ alkalmazott attribútum előfordulások és legyen $f \in FUNC\text{-}SET, f: \tau_1 \times \ldots \times \tau_l \rightarrow \tau_0$ függvény, és $\tau_i = TYPE(X_{pk_i}.a_i)$, $\tau_0 = TYPE(a)$, akkor van egy oszlop az $\{f(X_{pk_1}.a_1, X_{pk_2}.a_2, \ldots, X_{pk_l}.a_l) == a\}$, reláció számára az $\mathcal{F}_{UF}$-ben.
8. Legyen $\{c_1, c_2, \ldots, c_m\}$ különböző konstansok, amelyek előfordulnak a táblázat $target(e)$ oszlopában valamint az $\mathcal{U}$-ban. Akkor minden egyes c_i konstansra, amennyiben $TYPE(a) = TYPE(c_i)$, van egy $\{a == c_i\}$ reláció az $\mathcal{F}_{UCF}$-ben $(1 \quad i \quad m)$.

A továbbiakban egy példa következik arra, hogy a 4.16. ábrán bemutatott attribútum nyelvtan 3-as számú átírási szabályában (`Term`$_0$ → `Term`$_1$ `MulOp Factor`) a `Term`$_0$`.mode` attribútumhoz tartozó szemantikus függvény tanulásához milyen `T(Term`$_0$`.mode)` táblázat készíthető. Ez a módszer az attribútum nyelvtan más szabályában szereplő szemantikus egyenletek meghatározására is átvihető.

A szabályban található alkalmazott attribútum előfordulások `Term`$_1$`.mode`, `MulOp.operator`, `Factor.mode` jelölik a `Term`$_1$ nemterminálisból levezethető kifejezés típusát, a multiplikatív operátor típusát és a `Factor` nemterminálisból levezethető kifejezés típusát. Mivel a szabály rekurzív, ezért a `Term`$_1$`.mode` attribútum példányok értékeit a bölcstől kell megkérdezni a rendszernek egy adott input esetén. Az U_1 oszlopban "*" jelzi a *bölcstől* kapott attribútum értékeket. Ebben a példában $\mathcal{F}_{UR} = \emptyset$, mivel a *FUNC-SET*-ben nem voltak logikai értékű függvények.

class	*word*	*target*	$\mathcal{U}$			$\mathcal{F}_{UCR}$						$\mathcal{F}_{UF}$		$\mathcal{F}_{UCF}$	
		$T_0.mode$	U_1	U_2	U_3	R_1	R_2	R_3	R_4	R_5	R_6	F_1	F_2	C_1	C_2
+	3 × 2,5	*real*	*int**	*mul*	*real*	T	F	T	F	F	T	F	T	F	T
+	5 × 3	*int*	*int**	*mul*	*int*	T	F	T	F	T	F	T	F	T	T
+	1,5 × 4	*real*	*real**	*mul*	*int*	F	T	T	F	T	F	F	T	T	F
+	2,5 / 3	*real*	*real**	*div*	*int*	F	T	F	T	T	F	F	T	T	F
+	2 / 3	*real*	*int**	*div*	*int*	T	F	F	T	T	F	F	T	F	F
+	6 / 3,4	*real*	*int**	*div*	*real*	T	F	F	T	F	T	F	T	F	T
-	2 × 3,2	*int*	*int**	*mul*	*real*	T	F	T	F	F	T	T	F	T	F
-	4,3 / 2	*int*	*real**	*div*	*int*	F	T	F	T	T	F	T	F	F	T
-	8 / 3	*int*	*int**	*div*	*int*	T	F	F	T	T	F	T	F	T	T

A táblázatban a következő oszlopok találhatók:

U_1: $Term_1$.mode
U_2: MulOp.operator
U_3: Factor.mode
R_1: {$Term_1$.mode == int}
R_2: {$Term_1$.mode == real}
R_3: {MulOp.operator == mul}
R_4: {MulOp.operator == div}
R_5: {Factor.mode == int}
R_6: {Factor.mode == real}
F_1: {$Term_0$.mode == int}
F_2: {$Term_0$.mode == real}
C_1: {$Term_0$.mode == Term1.mode}
C_2: {$Term_0$.mode == Factor.mode}
T: igaz (true)
F: hamis (false)

4.18. Ábra. Egy propozíciós tanulási táblázat

Amennyiben egy attribútum érték tanuló algoritmus meg tudja találni a megoldást pl. a

$\oplus_1$: R_1 = true **and** R_3 = true **and** R_5 = true **and** F_1 = true
$\oplus_2$: F_2 = true

formulákat. A végső lépésben a ezeket a formulákat szemantikus függvényekké kell alakítani. Az *R*(3) átalakított szemantikus függvénye a következő alakú:

```
3: Term0 → Term1 MulOp Factor
   R(3): if Term1.mode == int and MulOp.operator == mul and Factor.mode == int
         then
            Term0.mode := int            ⊕1 alapján
         else
            Term0.mode := real           ⊕2 alapján
```

ami egy korrekt megoldás.

Az itt bemutatott példa egy S-attribútumos nyelvtan szemantikus függvényének tanulását mutatta be. A magyar természetes nyelvi egyértelműsítést végző alkalmazásban is egy S-attribútumos attribútum nyelvtan fog szerepelni. Az L-attribútumos nyelvtanok szemantikus függvényeinek tanulásával részletesen foglalkoznak a [Gyim97] publikációban.

4.6.3. A magyar nyelvi szófaji egyértelműsítés problémája

Az AGLEARN algoritmus alábbi alkalmazásában, a magyar szófaji egyértelműsítési probléma megoldása során ún. „removable" predikátum tanulása történik. Először ezt a megközelítést Cussens [Cuss97] alkalmazta az angol nyelv egyértelműsítésére. Ez azt jelenti, hogy amikor egy szónak több lehetséges morfológiai címkéje van, akkor bizonyos feltételek fennállása esetén ezekből néhányat (esetleg egy kivételével mindet) kizárhatunk. Egy ilyen predikátumot például tanulhatunk a C 4.5 tanuló algoritmussal, amelyhez a tréning adatokat a 4.5. Fejezetben ismertetett magyar korpusz szolgáltatja. Egy ilyen predikátum azt fogalmazza meg, hogy például a `PSN` címke kizárható egy bizonytalansági osztály előfordulás esetén, amennyiben egy a szomszédos szavak kódjainak segítségével megfogalmazott feltétel fennáll.

Az AGLEARN algoritmus a C 4.5 tanuló rendszerhez állít elő kibővített tanuló állományokat. Ennek alapján kézenfekvő az AGLEARN algoritmus hatékonyságát úgy vizsgálni, hogy a C 4.5 rendszert a bővítés nélküli tréning állományokra is lefuttatják. Az itt ismertetésre kerülő alkalmazás esetén is ez történt. Az eredmények a későbbiekben kerülnek ismertetésre.

A C 4.5 tanuló rendszerben a szövegkörnyezetet a szomszédos 7–7 szóra szorították meg. Amennyiben a környezet nem volt ennyi szó a bal vagy a jobb oldali környezetben, akkor egy speciális üres szimbólumot használtak az adatfájlok elkészítésekor. A környezet későbbi szűkítése lehetséges volt újabb adatfájl készítése

nélkül is, a C 4.5 rendszer adatfájljaiban egyes oszlopok figyelmen kívül hagyásával. A felhasznált korpusz tulajdonságai a 4.5. Fejezetben, a gyakoribb bizonytalansági osztályok pedig a 4.14. táblázatban találhatók.

A leggyakoribb 27 bizonytalansági osztály 6667 bizonytalanságot tartalmaz a 7229-ből (92,22%) a tréning adatok esetén, míg 2443 esetet tartalmaz a 2677-ből (91,60%) a teszt adatok esetén. A megtanult egyértelműsítő program annyi bizonytalanságot old fel, amennyit csak tud. Ahol nincs lehetőség a feloldásra, ott a bizonytalanságot a program meghagyja. Egy későbbi fázisban vagy valamilyen más módszer, pl. valószínűségszámítási modell alapján lesz lehetőség a bizonytalanságok teljes kiküszöbölésére, vagy a szintaktikus elemző fog bizonytalan input token sorozatot kapni. A legtöbb bizonytalansági osztály csak néhány (1–10) szót tartalmaz. A bizonytalansági osztályok között van néhány (3–4) kiemelkedően nagy elemszámú.

A C 4.5 tanuló rendszer számára az input fájlt a Orwell korpusz Prolog formájából készítették. A `T` osztály a CTAG kódolás szerint a névelőket jelenti. Az MSD rendszerben a névelőknek két részosztálya van: a határozott (`Tf`) és a határozatlan (`Ti`) névelők. A magyar nyelvben három névelő van: az *a*, *az* és az *egy*. Az utóbbi a határozatlan névelő. Az első kettő a határozott. A `[psn, t]` bizonytalansági osztályban egyetlen szó található: az *az*. A következőben ennek az osztálynak a segítségével mutatjuk be a tanulási folyamatot.

```
s('Ohu.1.2.6.3', [(ms, [ms, t]), asn, nso, nsnx, wpunct, (t, [psn, t]),
nsn, vmis3s, asn, npox, aso, t, asn, nsn, st, spunct]).
```

Minden esetben, amikor a `[psn, t]` bizonytalansági osztály egy képviselője előfordul egy mondatban, egy tréning példa keletkezik. Az itt látható mondatban a hatodik szó tartozik a kiválasztott osztályba, mivel a címkéi éppen `PSN` vagy `T` lehetnek. A C 4.5 példafájl egy részlete, amely a fenti esetet is tartalmazza itt látható:

```
...
Ohu.1.2.5.9, xxx, xxx, xxx, xxx, xxx, xxx, xxx,
                            nsd, pso, t, nso, vmis3s, vmn, wpunct, c_t
Ohu.1.2.6.3, wpunct, nsnx, nso, asn, ms, xxx, xxx,
                            nsn, vmis3s, asn, npox, aso, t, asn, c_t
Ohu.1.2.6.4, wpunct, nsnx, nsn, asn, t, wpunct, nsn,
                            asn, nsny, wpunct, psn, nsn, nsox, t, c_t
...
```

A C 4.5 input fájl lényegében egy táblázat. Minden egyes tréning példa egy sor, az egyes attribútumok pedig vesszővel elválasztva következnek egymás után. Az első attribútum a mondat azonosító, az utolsó pedig a kiválasztandó helyes címke (itt `c_t`), azaz a `T` tag. A tréning adatbázisban előforduló minden (1867) esetből egy tréning példa keletkezik. Az adatfájlban szerepel még 14 további attribútum: a mondatkörnyezet. Az első 7 a bal oldali, a második 7 érték a jobb oldali környezetben található szavak kódját tartalmazza. Az `xxx` kód a hiányzó tokeneket jelenti. A tanulás eredményeként kapott szabály a 4.19. ábrán láthatók, a tesztelés során kapott pontossági adatok pedig 4.20. táblázatban találhatók.

4.6.4. Az AGLEARN módszer alkalmazása az egyértelműsítési problémára

Az előző fejezetben a C 4.5 tanuló rendszer számára készített tréning fájlokban a morfológiai elemzés során kapott CTAG kódok írták le a többjelentésű szó környezetét. A C 4.5 rendszer olyan szabályokat állított elő a `[psn, t]` bizonytalansági osztályra, amelyekben a szomszédos szavak CTAG kódjai szerepeltek.

```
if ((token1_after == 'cp') ||
   ((token1_before == 'wpunct') && (token1_after == 'rg')) ||
   ((token1_before == 'xxx') && (token1_after == 'rg')) ||
    (token1_after == 'spunct') ||
   ((token1_before == 'wpunct') && (token1_after == 'st')) ||
    (token1_after == 't') ||
    (token1_after == 'vmcp3s') ||
   ((token1_before == 'wpunct') && (token1_after == 'vmip3s')) ||
    (token1_after == 'vmis3s') ||
    (token1_after == 'vpunct')) then Class = c_psn else Class = c_t
```

4.19. Táblázat. A C 4.5 tanuló rendszerrel kapott szabály a `[psn, t]` bizonytalansági osztályra

A tanulási alkalmazás készítése során az AGLEARN módszerhez a szerző és munkatársai egy egyszerű

nyelvészeti háttértudást fejlesztettek ki. Ennek a célja az volt, hogy szócsoportokat vagy jellegzetes szószerkezeteket (*szintagmákat*) próbáljanak meg felismerni a tárgyszó környezetében. Egy attribútum nyelvtant használtak a két új attribútum (a *kategória* és a *szintagma*) kiszámítására. A kategória attribútumot a bal és a jobboldali, a szintagma attribútumot értelemszerűen csak a jobboldali környezetre számították ki. Az utóbbi attribútum értéke a környezet szintaxisán alapult, ezért nem lehetett kiszámítani a baloldali környezetre, ugyanis ebben az esetben a szószerkezet kezdete nem határozható meg. Az AGLEARN egy C 4.5 input fájlt generált, amelyben a környezeteken kívül az újonnan generált attribútum értékek is szerepeltek, mint új oszlopok. A megtanult szabályokban néhány esetben szerepelnek is ezek az új értékek [Emde96].

Ablak méret	Megtanult szabályok mérete	Pontosság			
		A tréning adatokon (1867)		A teszt adatokon (620)	
		%	A hibák száma	%	A hibák száma
1	25	98,50	17	97,30	28
2	21	98,90	15	97,60	35
3	20	98,90	16	97,40	34
4	18	99,00	16	97,40	35
5	18	99,00	16	97,40	33
6	20	98,90	17	97,30	32
7	20	98,90	17	97,30	32

4.20. Táblázat. A C 4.5 tanuló rendszerrel kapott eredmények pontossága különböző ablak méretek esetén

Az attribútum nyelvtan a C 4.5 számára generált input fájlt használta a környezetek meghatározására. A kezdőszimbólum a `Sentences` volt, mint azt az alábbi példa mutatja (az attribútum nyelvtan egy nagyobb részlete megtalálható a 4.6.5. Fejezetben):

```
Sentences → Sentence_id "." BeforeCtags "," AfterCtags Sentences
Sentences → λ
```

A `[psn, t]` bizonytalansági osztály attribútum nyelvtanában a `Sentences` nemterminálisnak van egy attribútuma, amelynek a neve: `psn_or_t`. Ennek az attribútumnak két lehetséges értéke van: `psn` vagy `t`, így az AGLEARN módszerrel olyan szabályokat (szemantikus egyenleteket) tudtunk tanulni, amelyek a helyes szófaji címkéket adják meg, azokban az esetekben, amikor a mondatban a `[psn, t]` bizonytalanság fordul elő. A `BeforeCtags` nemterminális a bizonytalan jelentésű szó baloldali környezetét jelöli. Jelen esetben csak a tárgyszót közvetlenül megelőző tokent vizsgálták, a többit kihagyták a vizsgálatból. A szavak CTAG kódjait csoportokba sorolták a mondatokban elfoglalt szerepük szerint. A háttér attribútum nyelvtan ezt az értéket – a `BeforeCtags` nemterminális *kategóriáját* – számította ki. Ugyanezen a módon számította ki az `AfterCtags` nemterminális *kategóriáját* is. Ezen kívül a jobboldali környezetben szereplő CTAG értékekből határozták meg az `AfterCtags` nemterminális *szintagma* attribútumának az értékét. Az alábbi 4.21. táblázatban láthatók a `Sentences` nemterminálishoz tartozó átírási szabályokban szereplő attribútumok.

A generált kibővített táblázatban még két további logikai érték is szerepelt, mint két újabb oszlop:

```
BeforeCtags.before1_ctag == AfterCtags.after1_ctag
BeforeCtags.group == AfterCtags.group
```

Attribútum név	Típus	Magyarázat
BeforeCtags.before1_ctag	CTAG	A tárgyszó előtt álló szó CTAG-e
BeforeCtags.group	GROUP	A tárgyszó előtt álló szó, kategóriája
AfterCtags.after1_ctag	CTAG	A tárgyszó utáni szó CTAG-e
AfterCtags.group	GROUP	A tárgyszó után álló szó, kategóriája
AfterCtags.syntagma	SYNTAGMA	A tárgyszó jobboldali környezetéből számított mondat részre utaló attribútum

4.21. Táblázat Az AGLEARN módszer által használt fontosabb attribútumok

Minden egyes tréning példára egy sor generálódik a C 4.5 input fájlba. Ezt a fájlt használja a C 4.5, hogy például egyértelműsítési szabályokat generáljon a `[psn, t]` bizonytalansági osztályra. A megtanult

szabályokból előállítható az a szemantikus függvény amely ki tudja számítani a Sentences nemterminális *psn_or_t* attribútumának az értékét. A C 4.5 adatfájl egy részlete látható a 4.22. ábrán.

```
...
Ohu.1.2.5.7,  rg, Oth, cp, Oth, AttSynt, false, true, c_psn
Ohu.1.2.14.1, cp, Oth, t, Oth, SubjSynt, false, true, c_psn
Ohu.1.2.14.2, wpunct, Oth, t, Oth, noneSynt, false, true, c_psn
Ohu.1.2.15.1, wpunct, Oth, vmis3s, Pred, AccSynt, false, false, c_psn
...
Ohu.1.2.1.1,  wpunct, Oth, npn, Subj, SubjSynt, false, false, c_t
Ohu.1.2.2.1,  xxx, none, nso, AdvOth, noneSynt, false, false, c_t
Ohu.1.2.2.6,  cp, Oth, nsn, Subj, SubjSynt, false, false, c_t
Ohu.1.2.2.9,  nsax, Acc, asn, Att, AttSynt, false, false, c_t
Ohu.1.2.2.10, vmip3s, Pred, nsa, Acc, AccSynt, false, false, c_t
...
```

4.22. Ábra Az AGLEARN által előállított C 4.5 adatfájl

A megtanult döntési fából a következő szemantikus egyenlet állítható elő:

```
if ((after1_ctag == 't') ||
    (after1_ctag == 'wpunct') ||
    (after1_ctag == 'spunct') ||
    (after1_ctag == 'cp') ||
   ((before1_ctag == 'rg') && (after1_group == Oth) &&
    (after1_syntagma == AdvOthSynt)) ||
   ((before1_ctag == 'wpunct') && (after1_group == Att) &&
    (after1_syntagma == noneSynt)) ||
    (after1_group == Pred))
then
  psn_or_t = c_psn ;
else
  psn_or_t = c_t ;
```

4.23. Ábra Az AGLEARN által tanult szemantikus egyenlet a [psn, t] osztályra

Bizonytalansági osztály	C 4.5 eredményei				Az AGLEARN eredményei				
	Tréning		Teszt példák		Tréning		Teszt példák		
	#err	%	#err	%	#err	%	#err	%	Jelzés
[asn, vmis3s]	40	8,2	18	9,9	40	8,2	18	9,9	-
[cp, rg, vmip3s]	14	5,7	31	24,8	14	5,7	31	24,8	-
[cp, rg]	142	16,8	74	25,2	141	16,0	70	23,8	+
[cp, rp]	41	12,3	16	10,7	19	5,7	8	5,4	++
[cp, vmis3s]	2	1,8	0	0,0	2	1,8	0	0	-
[nsn, psn]	24	21,6	16	30,8	2	1,8	5	9,6	++
[psn, rp]	9	6,3	3	5,3	9	6,3	3	5,3	-
[psn, t]	25	1,5	17	2,7	21	1,1	16	2,6	+
[pso, rg]	73	33,6	34	40,0	35	16,1	19	22,0	++
[rg, rp]	57	38,0	15	25,4	57	38,0	15	25,4	-
[rg, st]	104	36,5	44	44,0	84	29,5	40	40,0	++

4.24. Táblázat A C 4.5 és az AGLEARN által kapott eredmények összehasonlítása

A 4.24. Táblázatban a csak C 4.5-tel és az AGLEARN-nel kapott eredmények láthatók. A táblázatban a legnépesebb, legalább 100 előfordulást tartalmazó bizonytalansági osztályok eredményei találhatók. A Jelzés oszlopban található - jel azokat az osztályokat jelöli, ahol az eredmények azonosak lettek. A + jel kis mértékű javulást, a ++ jel nagyobb mértékű javulást jelöl. A -- jel a rosszabbodást jelentette volna, de ilyen eset nem volt. Azt a következtetést vonhatjuk le, hogy az AGLEARN alkalmazásával a tanult szabályok pontossága növelhető. Ennek az, az oka, hogy az AGLEARN fel tudja használni azt a háttértudást, amely az attribútum nyelvtanban testesül meg. Megjegyezzük, hogy az előző részben ismertetett attribútum nyelvtan más bizonytalansági osztályok esetén is alkalmazható, vagyis nem kell minden bizonytalansági osztályhoz új attribútum nyelvtant kifejleszteni. Ehhez az szükséges, hogy a Sentences nemterminális attribútumának a

nevét (a korábban példaként említett esetben psn_or_t volt) és a tárgyalt bizonytalansági osztályhoz tartozó környezeteket tartalmazó input fájl nevét megváltoztassák. Ezen kívül, ugyanez az attribútum nyelvtan használható végső egyértelműsítő eszközként is, amint a megtanult szemantikus egyenleteket visszaírjuk az attribútum nyelvtanba. A kibővített C 4.5 input fájl készítését támogató részeket pedig el kell távolítani. A munka folyatatásaként lehetőség van a szabályok további finomítására kézzel, nyelvész szakértő(k) igénybe vételével, illetve nagyobb tréning adatbázis felhasználására.

4.6.5. Az AGLEARN módszerhez kifejlesztett attribútum nyelvtan egy részlete

```
Sentences = Sentence_ID "," BeforeCtags "," AfterCtags Sentences ;
do
end

Sentences = ;
do
end
...

AfterCtags = Acc_Group "," Synt_Acc;
do
    syntagma = Synt_Acc.syntagma;
    ctag= Acc_Group.ctag;
    group = Acc;
end

AfterCtags = AdvDat_Group "," Synt_AdvDat;
do
    syntagma = Synt_AdvDat.syntagma;
    ctag= AdvDat_Group.ctag;
    group = AdvDat;
end

AfterCtags = AdvOth_Group "," Synt_AdvOth;
do
    syntagma = Synt_AdvOth.syntagma;
    ctag= AdvOth_Group.ctag;
    group = AdvOth;
end
...

Synt_Acc = Pred_Group "," Ctags;
do
    syntagma= AccSynt;
end

Synt_Acc = NonPred_Group "," Synt_Acc;
do
    syntagma= Synt_Acc.syntagma;
end

Synt_Acc = Class_ID;
do
    syntagma= noneSynt;
end

Synt_AdvDat = Pred_Group "," Ctags;
do
    syntagma= AdvDatSynt;
end

Synt_AdvDat = NonPred_Group "," Synt_AdvDat;
do
    syntagma= Synt_AdvDat.syntagma;
end

Synt_AdvDat = Class_ID;
do
```

```
    syntagma= noneSynt;
end

Synt_AdvOth = Pred_Group "," Ctags;
do
    syntagma= AdvOthSynt;
end

Synt_AdvOth = NonPred_Group "," Synt_AdvOth;
do
    syntagma= Synt_AdvOth.syntagma;
end

Synt_AdvOth = Class_ID;
do
    syntagma= noneSynt;
end

Synt_Subj = Pred_Group "," Ctags;
do
    syntagma= SubjSynt;
end

Synt_Subj = NonPred_Group "," Synt_Subj;
do
    syntagma= Synt_Subj.syntagma;
end

Synt_Subj = Class_ID;
do
    syntagma= noneSynt;
end
...
```

Az attribútum nyelvtanban használt CTAG kód *kategóriák*:

```
% group of ctags which might be the subject of the sentence
Subj_Group =
   npn ,npnx,npny,nsn,nsnx,nsny,
   ppn,ppnx,ppny,psn,psnx,psny;

% group of ctags which might be the predicate of the sentence
Pred_Group =
   vmcp1s,vmcp1p,vmcp2,vmcp2s,vmcp2p,vmcp3s,vmcp3p,
   vmip1s,vmip1p,vmip2,vmip2p,vmip2s,vmip3s,vmip3p,
   vmis1s,vmis1p,vmis2,vmis2p,vmis2s,vmis3s,vmis3p,
   vmmp1s,vmmp1p,vmmp2,vmmp2p,vmmp2s,vmmp3s,vmmp3p,
   va;

% group of ctags which might be the accusative of the sentence
Acc_Group =
   vmn,apa,apax,apay,asa,asax,asay,
   npa,npax,npay,nsa,nsax,nsay,
   ppa,ppax,ppay,psa,psax,psay;

% group of ctags which might be the dative adverb of the sentence
AdvDat_Group =
   apd,apdx,apdy,asd,asdx,asdy,
   npd,npdx,npdy,nsd,nsdx,nsdy,
   ppd,ppdx,ppdy,psd,psdx,psdy;

% group of ctags which might be the other adverb of the sentence
AdvOth_Group =
   apo,apox,apoy,aso,asox,asoy,
   npo,npox,npoy,nso,nsox,nsoy,
   ppo,ppox,ppoy,pso,psox,psoy;

% group of ctags which might be the attribute of the sentence
Att_Group =
```

```
    apn,apnx,apny,asn,asnx,asny,
    mp,mpx,mpy,ms,msx,msy;

% group of other ctags in the sentence
Oth_Group
    md,i,cp,cpunct,spunct,wpunct,
    rg,ro,rp,rq,rv,st, t,
    unknown,x,y,xxx;

Groups =  Subj, Pred, Acc, AdvDat, AdvOth, Att, Oth, none ;

Syntagmas = SubjSynt, PredSynt, AccSynt, AdvDatSynt,
     AdvOthSynt, AttSynt, OthSynt, noneSynt ;
```

4.7. A szófaji egyértelműsítési probléma megoldása különböző tanuló algoritmusokkal

Az alábbiakban a magyar nyelvi szófaji egyértelműsítési probléma tanuló algoritmusokkal történő megoldásának lehetőségei kerülnek ismertetésre. A 4.7. fejezet ugyanebben a formában angol nyelven megjelent a [Hor99] kiadványban. A cikkben a szerzők 5 különböző tanuló algoritmust vizsgáltak meg; a C 4.5, az AGLEARN és a Progol rendszert Szegeden (Alexin Zoltán és munkatársai), míg a RIBL és a PHM módszert Bonnban (Horváth Tamás és Stefan Wrobel). A kísérletekhez a 4.5. Fejezetben ismertetett annotált magyar korpuszt használták fel, az ún. Orwell korpuszt [TELRI], amely mintegy 100 000 tokenből áll (szövegszó és írásjel együtt). Ez az anyag viszonylag kis méretű összehasonlítva a 3 millió szó méretű angol Wall Street Journal korpusszal vagy a 2 millió szavas svéd UMEÅ korpusszal.

Az esettanulmányban felhasználták az MTA-SZTE Mesterséges Intelligencia Kutató Csoport által kifejlesztett AGLEARN tanuló algoritmust. Az AGLEARN módszerről és alkalmazásáról a szófaji egyértelműsítési problémára a 4.6. Fejezet tartalmazott részletes ismertetést. Az AGLEARN algoritmus feladata egy attribútum nyelvtan átírási szabályaihoz rendelt szemantikus függvények tanulása. A tanulásban az attribútum nyelvtan (*környezet-független*) átírási szabályai, háttér (*segéd*) szemantikus függvények, valamint egy tréning adatbázis vesz részt. Ez utóbbiban pozitív és negatív tanulási példák találhatók.

A szófaji egyértelműsítés a számítógépes természetes nyelv feldolgozási feladatok (úgy, mint elemzés, információgyűjtés stb.) egyik első lépése. Az egyértelműsítés feladata egy adott szöveg minden egyes szavához egy egyértelmű szófaji besorolást (*címkét*) kiszámítani, amely az adott szó morfológiai kategóriáját reprezentálja. A lehetséges kategóriákat nyelvészek határozzák meg, az adott természetes nyelv és a konkrét alkalmazás igényeinek megfelelően. Az egyértelműsítés (*tagging*) azért nehéz feladat, mert a természetes nyelvekben nagyszámú bizonytalanság fordul elő (a magyar nyelvben minden negyedik-ötödik szó többértelmű). A finn-ugor nyelvekben, mint amilyen a magyar nyelv is, ez a feladat még nehezebb, mert gazdag morfológiával rendelkeznek, és nagymértékben szabad a mondatok szórendje. Bár egy redukált *tag* (morfológiai kategória) halmaz[44] segít abban, hogy kiküszöböljük a nagyszámú morfológiai kategória okozta hátrányokat, azonban a szabad szórend még így is egy újabb kihívást jelent a tanuló algoritmusok számára.

A 4.7.1. Fejezetben egy rövid nyelvészeti bevezetés található, valamint ismertetés a felhasznált korpuszról. Ebben a fejezetben található egy összefoglaló is arról, hogy más európai tudományos műhelyekben milyen eredményeket értek el a szófaji egyértelműsítés területén.[45] A 4.7.2. Fejezetben az AGLEARN algoritmus rövid áttekintése található. A 4.7.3. Fejezetben a kiválasztott öt tanuló algoritmussal végzett kísérlet tervezése található.

Az 4.7.4. – 4.7.8. Fejezetekben a különböző tanuló algoritmusok alkalmazásának eredményei találhatók. A kísérleti eredmények azt mutatták, hogy a magyar szófaji egyértelműsítési probléma valóban nem könnyű feladat, és még a legegyszerűbb háttér tudás hozzáadása is javít a megtanult szabályrendszer pontosságán. A kapott eredmények alapján a példány-alapú tanulás[46] is egy ígéretes megközelítésnek tűnik.

---

[44] A szóban forgó kategória halmazt CTAG-nek (Corpus TAG) nevezik és az MTA Nyelvtudományi Intézetében alakították ki a koncepcióját [27].

[45] Természetesen ezekben az esetekben országonként más és más természetes nyelv képezte a kutatások tárgyát.

[46] Instance Based Learning

A tanulmány 4.7.11. Fejezetében a különböző tanuló algoritmusokkal kapott szabályrendszerek (Prolog egyértelműsítő programok) kaszkád (*soros*) kapcsolását vizsgáltuk meg. Az eredmények azt mutatják, hogy két algoritmus megfelelő összekapcsolásával elérhető, hogy a kapott egyesített algoritmus pontossága mindkét korábbi algoritmus pontosságánál nagyobb legyen.

4.7.1. A tanuló algoritmusokban alkalmazott Prolog korpusz formátum

A tanuló algoritmusok alkalmazásához a magyar korpusz Prolog formáját használtuk. A konverzió mondatonként történt és az SGML fájlban lévő <s> és </s> mondathatároló jelek közti szöveget egy HuMor alapú elemző programmal dolgoztuk fel. A kapott fájlt a `dis2pl.exe` nevű konvertáló programmal alakítottuk át Prolog formájúvá.

```
s('Ohu.1.2.1.1', [('Afp-sn', ['Afp-sn', 'Vmis3s---n']), 'WPUNCT', 'Afp-sn', 'Afp-
sn', 'Nc-sn', ('Vmis3s---n', ['Afp-sn', 'Nc-sn', 'Vmis3s---n']), 'WPUNCT', ('Tf',
['P---sn', 'Tf']), 'Nc-pn', 'Rg', 'Mc-sa', ('Vmis3p---n', ['Afp-pn', 'Vmis3p---
n']), 'SPUNCT']).
s('Ohu.1.2.1.2', ['Nc-sn', 'Nc-sn', 'WPUNCT', 'Nc-sa---s3', 'Rv', 'WPUNCT', 'Afp-
sw', ('Vmis3s---n', ['Afp-sn', 'Vmis3s---n']), 'Tf', 'Nc-sn', 'Nc-sp---s3',
'WPUNCT', 'Ccsw', 'Vmmp3s---n', 'Tf', 'Afp-sn', ('Nc-sb', ['Nc-sb', 'Rg']),
'SPUNCT']).
s('Ohu.1.2.1.3', [('Ccsw', ['Ccsw', 'Rg']), ('Rg', ['Nc-sn', 'Rg']), ('Vmis3s---n',
['Afp-sn', 'Vmis3s---n']), 'P---sn', 'Afp-sw', 'Vmn', 'WPUNCT', 'Ccsw', 'Rg',
'Vmmp3s---n', 'Rp', 'P---si', ('Rg', ['Rg', 'Rp']), ('Ti', ['Mc-sn', 'Ti']), 'Afp-
sn', 'Nc-sn', 'SPUNCT']).
s('Ohu.1.2.2.1', [('Tf', ['P---sn', 'Tf']), 'Nc-st', ('Afp-sn', ['Afp-sn', 'Nc-
sn']), 'Nc-sn', ('Afp-sn', ['Afp-sn', 'Vmis3s---n']), 'Nc-sn', 'Ccsw', 'Afp-sn',
'Nc-sn', ('Nc-pn', ['Afp-pn', 'Nc-pn']), 'Nc-sa---s3', 'Vmis3s---y', 'SPUNCT']).
...
```

A mondatkezdő <s id='Ohu.1.2.1.1'> *tag*ben megadott azonosítót mind az elemző program, mind a Prologra konvertáló program továbbviszi, és a Prolog korpuszban is ez lesz a tekintett mondat azonosítója. Az Orwell regényben a tréning adatbázisként használt első két fejezetben 4583 mondat (71519 token), a teszt adatokként használt harmadik és negyedik részben pedig 2185 mondat (26908 token), összesen 6768 mondat (98427 token) volt. A tanuló programokkal előállított egyértelműsítő szabályok (Prolog programok) szavankénti pontosságát úgy számítottuk ki, hogy a teszt adatbázison végrehajtottuk az egyértelműsítést, majd az eredményt összevetettük a kézi egyértelműsítés eredményével. A továbbiakban a tanuló rendszerek inputját Prolog programokkal[47] állítottuk elő a korpusz Prolog formájának felhasználásával.

4.7.2. Az AGLEARN tanuló algoritmus

Az AGLEARN algoritmus (lásd részletesen a 4.6. Fejezetben) segítségével ún. környezeti szabályok határozhatók meg a helyes annotáció meghatározására. Használatával a C 4.5 döntési-fa tanuló rendszer számára állítható elő egy input fájl. Az AGLEARN segítségével nagyobb pontosság érhető el, mint a C 4.5 alkalmazásával egyedül, mivel a generált C 4.5 input fájl nemcsak a szövegkörnyezetről, hanem a mondat szerkezetéről is tartalmazott információkat. A mondatszerkezet alapján kiszámított attribútumokat egy (*háttér*) attribútum nyelvtan segítségével definiálták. Ennek az attribútum nyelvtannak a használatával a C 4.5 pontosabb egyértelműsítési szabályokat határozott meg.

Az ILP-ben a logikai programozási környezet nyújtja az eszközt a tanulási példák, a háttértudás, illetve a megtanulandó fogalmak definíciójához. Az AGLEARN eljárásban a háttértudást és a fogalmakat egy attribútum nyelvtan írja le. A fejlesztő által kidolgozott attribútum nyelvtan szintaxis és az adott háttér tudás együtt lecsökkentheti fogalomnak a keresési terét és gyorsabb, valamint egyszerűbb reprezentációt adhat. Az AGLEARN algoritmusban egyelőre S-attribútumos és L-attribútumos attribútum nyelvtanok használhatók és egyszerű szemantikus egyenletek tanulhatók. Megjegyezzük azonban, hogy háttértudásként ennél lényegesen komplexebb attribútum nyelvtanok is használhatók például OAG [Kas80].

---

[47] A SICStus Prolog 3.7.1. rendszert használtuk a programozási munkákhoz, illetve az azóta folyamatosan érkező újabb upgrade verziókat.

Ebben a fejezetben az AGLEARN módszer alkalmazását ismertetjük a magyar mondatok szófaji egyértelműsítésére. A helyes jelentés kiválasztására környezeti szabályokat határoztunk meg a tanuló algoritmus felhasználásával. A 100 000 token méretű TELRI korpusz [TELRI] egy része biztosította a tréning adatokat, egy másik részlete pedig volt a teszt. Az AGLEARN segítségével a C 4.5 [Quin93] döntési-fa tanuló rendszer számára állítottunk elő adatokat. Az attribútum nyelvtan alkalmazásával generált kiegészítő adatsorozat a mondatok szerkezetével kapcsolatos információkat tartalmazott. Az attribútum nyelvtan a többjelentésű szó környezetének szintaxisát írta le, és a mondatstruktúrával kapcsolatos szintetizált attribútum értékeket számított ki. Az AGLEARN alkalmazásával sikerült pontosabb szabályrendszert előállítani ahhoz az esethez képest, amikor a C 4.5 rendszert csak önmagában alkalmaznánk.

4.7.3. A tanuló algoritmusok alkalmazása és az eredmények kiértékelésének megtervezése

Ebben a fejezetben öt kiválasztott tanuló algoritmussal végzett kísérletek eredményét foglaljuk össze, amelyeket a magyar szófaji egyértelműsítési probléma megoldására végeztünk. A munka részben a német GMD intézetben illetve Magyarországon folyt. A cél nem csupán az egyes rendszerek összehasonlítása volt, hanem az is, hogy megvizsgáljuk a magyar nyelv szokatlan tulajdonságainak (sok MSD kód, szabad szórend) hatását, a tanulási feladatra. Ezért öt különböző elvi alapon működő tanuló algoritmust választottunk ki, hogy ezen a feladaton kipróbáljuk őket és megvizsgáljuk a teljesítményüket. Referenciaként két jól ismert standard tanuló-algoritmust választottunk ki: a C 4.5 döntési-fa tanuló rendszert [Quin93] és a Progol ILP rendszert [Mugg95].

Mivel más nyelvek esetében jó eredményt értek el a memória alapú (memory-based) módszerekkel, ezért egy ilyen algoritmust is választottunk a RIBL-t [BohnT], [Emde96] összehasonlításul. A háttértudás fontosságának igazolására, amelyről már Cussens is említést tett [Cuss97], kísérleteket végeztünk az AGLEARN tanuló algoritmussal. Ez a rendszer további attribútum-értékeket tud kiszámítani a C 4.5 számára. Az új értékek egy attribútum-nyelvtan segítségével számíthatók ki. Az adott esetben egy elemző programot kell lefuttatni a többértelmű szó egy környezetén. Végül egy speciális színezett út-gráf háttértudást használó tanuló algoritmust választottunk ki, amely mélységi korlát nélküli szabályrendszert tud hatékonyan tanulni: PHM [Hor96], [HorPh].

Az alábbiakban vázoljuk az egyes reprezentációkat, amelyeket az egyes tanuló rendszerekben használtunk. Néhány általános tulajdonságot ismertetünk:

- először, eltérően a [Cuss97]-ben alkalmazott módszertől, ahol *„removable"* szabályokat tanultak (mi nem lehet egy tárgyszó címkéje), mi ún. *choose* szabályokat próbáltunk meg tanulni, azaz olyan szabályt hogy, amikor azt alkalmazzák, akkor a szabály által eredményül adott tag, a szabály által helyesnek ítélt eredmény tag legyen.
- másodszor, hogy növeljük a specifikusságát a rendszernek, és hogy csökkentsük a futási időt, elhatároztuk, hogy szétvágjuk a tanulási problémát és minden egyes bizonytalansági osztályra külön tanulunk szabályokat. A kísérletben nem használtuk fel az összes bizonytalansági osztályt, csak azokat, amelyet legalább 50-szer fordultak elő a tréning korpuszban. Ezek az osztályok az összes bizonytalanság 92.23%-át reprezentálták, 6667-t az összes 7229 bizonytalan szóból a tréning adatbázis esetén, illetve 91.26%-ot, 2443-t az összes 2677 bizonytalan szóból a tesz adatbázis esetén (részletes információ erről a 6. Táblázatban található). A C 4.5, az AGLEARN és a RIBL esetében külön tanulási problémát állítottunk elő minden bizonytalansági osztályra. A Progol és a PHM esetén külön tanulási problémát állítottunk elő minden choose_t_i_from_t_1_..._t_k predikátumra, minden egyes lehetséges *tag*re, és minden lehetséges c = {t_1, ..., t_k} bizonytalansági osztályra.

Minden példában a tárgyszó már egyértelműsített környezetét használtuk fel, az egyetlen helyes *tag*gel. A C4.5, az AGLEARN, a Progol és a PHM esetében a megtanult szabályok tesztelése a következőképpen folyt: ha pontosan egy egyértelműsítési szabály volt alkalmazható, akkor ezt a szabályt használtuk a helyes tag kiszámítására. Amennyiben egyetlen szabályt sem, vagy esetleg többet is lehetett volna alkalmazni, akkor ezt nem meghatározhatónak vettük, és mint [Cuss97]-ben, a tréning adatbázis alapján kiszámított leggyakoribb (legvalószínűbb) jelölttel helyettesítettük. Megjegyezzük, hogy egy szabály nem alkalmazható volt, amennyiben nem illeszkedett a tárgyszóra, vagy alkalmazásakor valamelyik környezetben szintén nem egyértelmű szó (szavak) fordult elő. Mivel az IBL paradigma képes kezelni a hiányzó attribútumokat is, ezért a RIBL esetében a többértelmű tokeneket olyan objektumoknak tekintettük, amelyeknek nem volt osztályba sorolási információja.

4.7.4. A C 4.5 tanuló algoritmus alkalmazása

A C 4.5-tel végzett kísérleteinkben a tárgyszó környezetét fix méretű token listával reprezentáltuk.

Esetünkben a választott *ablak* méret 7 volt, azaz 7 szó a kiválasztott többértelmű szó előtt és 7 szó utána. Kisebb ablakméret használatához a C 4.5 egy opcióját használhatjuk – egyes oszlopokat kihagyhatunk a tanulás során (*ignore*). Ezzel a módszerrel ugyanazt az adatfájlt használhatjuk 7 tokennél rövidebb környezetek esetén, csupán egy konfigurációs fájlt kell cserélni, amelyben a megfelelő oszlopokat a tanulásból kizárjuk.

Az alábbiakban a C 4.5 adatfájl egy sora látható:

```
empty, empty, ms, asn, nso, nsnx, wpunct, nsn, vmis3s, asn, npox, aso, t, asn, t
```

Az első 7-7 oszlop a bal oldali környezetet (`empty, empty, ms, asn, nso, nsnx, wpunct`) és a jobb oldali környezetet (`nsn, vmis3s, asn, npox, aso, t, asn`) írja le ebben a sorrendben, az utolsó oszlopban pedig a helyes tag (`t`) áll. Amennyiben a környezet rövidebb volt 7 tokennél, úgy egy speciális értékkel az `empty`-vel töltöttük fel.

A 4.7.5. Fejezetben ezt az egyszerű adatsort majd bizonyos nyelvészeti tudással is kiegészítjük az AGLEARN módszer segítségével, amely további attribútum értékeket (oszlopokat) számít ki. Az AGLEARN alapötlete nagyon hasonlít a LINUS [Dzer93] ILP rendszer alapötletéhez, nevezetesen: a relációs tanulási problémát propozícionális tanulási feladattá alakítjuk, majd a megtanult szabályokat visszaalakítjuk relációs formájúakká. A különbség az, hogy az AGLEARN attribútum nyelvtan formalizmust használ a relációk reprezentációjához.

4.7.5. Az AGLEARN tanuló algoritmus alkalmazása

Az AGLEARN tanuló rendszer alkalmazásáról egy részletes ismertetés található a 4.6.4. Fejezetben. Az alkalmazás készítése során a szerző és munkatársai egy egyszerű nyelvészeti háttértudást fejlesztettek ki. A háttér attribútum nyelvtannal szócsoportokat vagy jellegzetes szószerkezeteket (*szintagmákat*) próbáljanak meg felismerni a tárgyszó környezetében. Két új attribútumot (a *kategória* és a *szintagma*) vezettek be. A kategória attribútumot a bal és a jobboldali, a szintagma attribútumot értelemszerűen csak a jobboldali környezetre számították ki. Az utóbbi attribútum értéke a környezet szintaxisán alapult, ezért nem lehetett kiszámítani a baloldali környezetre, ugyanis ebben az esetben a szószerkezet kezdete nem határozható meg. Az AGLEARN egy C 4.5 input fájlt generált, amelyben a környezeteken kívül az újonnan generált attribútum értékek is szerepeltek, mint új oszlopok (attribútumok). A megtanult szabályokban néhány esetben megjelentek ezek az új értékek [Emde96].

4.7.6. A Progol tanuló algoritmus alkalmazása

Minden egyes $c = \{t_1, \ldots, t_k\}$ bizonytalansági osztályra és minden egyes $t_i \in c$ tokenre egy különálló tanulási feladatot készítettünk elő. Minden egyes c bizonytalansági osztály előfordulás esetén egy `r(L, R)` tényt adtunk a pozitív, illetve a negatív példák halmazához annak megfelelően, hogy a tréning adatbázis szerinti helyes címke t_i volt-e (ilyenkor a pozitív példákhoz adjuk) vagy nem (ekkor pedig a negatív példákhoz). Az `L` és az `R` argumentumok a bal és a jobboldali teljes környezetek (nincs korlát a környezet hosszára nézve) ebben sorrendben. A környezetek helyes szófaji címke listák. A bal oldali környezet meg van fordítva, hogy a lista feje a bal oldali szomszédos szó lehessen. Minden Progol tanulási feladatban egy technikai háttér tudást is felhasználtunk:

$$\cup \{t(t)\} \cup \{\texttt{empty([]),first(T, [T|\_]), second(T,[\_,T|\_]), third(T,[\_,\_,T|\_])}\},\ t \in c$$

ahol a `c` a bizonytalansági osztályt jelöli, a `first/2`, `second/2` és a `third/2` predikátumokat az első, második illetve a harmadik token kiválasztására használhatjuk. Ez a háttér tudás implicit módon ugyan de azért szintén korlátozza a figyelembe vett környezet méretét. A Progol eredményként az alábbihoz hasonló formulákat ad:

```
choose_cp_from_cp_rp(L,R) :- first(L1,L), rg(L1), second(R2,R), nsn(R2).
```

ami azt jelenti, hogy a szabály által ajánlott szófaji címke `cp` a `[cp, rp]` bizonytalansági osztály esetén, ha a baloldali környezet első CTAG-e (közvetlenül megelőző szó) `rg`, **és** a jobboldali környezet második tokene `nsn`.

4.7.7. A PHM tanuló algoritmus alkalmazása

A szorzat homomorfizmus módszer (PHM, Product Homomorphysm Method) [Hor96] egy kombinatorikus megközelítés egyszerű logikai programok tanulására. Felhasználva a PHM módszert, PAC[48] tanulással kapcsolatos pozitív tételek bizonyíthatók strukturált háttértudás osztályokra, mint amilyen például a színezett útvonal gráfok osztálya [Hor96], [Hor97], [HorPh]. A PHM módszer alkalmazásakor feltételezzük, hogy az egyértelműsítési probléma lényegében egy szekvenciális feladat, amelyet útvonal gráffal lehet reprezentálni. Mivel a PHM-ben nincs semmilyen megkötés a mélységre, ezért eltérően más módszerektől itt nincs szükség egy ablak kijelölésére (limitálni a figyelembe vett környezet méretét).

A színezett útvonal gráfokat egy $G = (V, E, Q_1, \ldots, Q_l)$, jelöli, ahol V jelöli a gráf csúcsainak a halmazát, $E \subseteq V \times V$, az gráf éleinek a halmaza és $Q_i \subseteq V$, minden $i = 1, \ldots, l$ a csúcsok színei. Egy csúcsnak így lehet több színe is, de az is lehet hogy nincs egyetlen egy sem. A G egy színezett útvonal gráf, ha a (V, E) irányított gráf irányított nem-összefüggő utak halmaza.

Ahhoz, hogy a PHM módszert alkalmazni lehessen a szófaji egyértelműsítési problémára, a háttértudást színezett útvonal gráf alakúra kell átalakítani. A kísérletben a PHM módszerhez a következő háttértudást használtuk, amelyet a tréning adatbázisból konstruáltunk meg a következőképpen:

- minden `i`-re és `j`-re, egy konstans került bevezetésre, amely az `i`-ik (`i` azonosítójú) mondat `j`-ik szavát jelöli
- minden egyes szomszédos `a` és `b` szóra, egy `r(a,b)` tényt adunk a háttértudáshoz, az ilyen `r(a,b)` tények végül nem-összefüggő utakat alkotnak a háttértudásban
- Minden a konstans esetén egy `t(a)` tényt adunk a háttértudáshoz, ahol a `t` az a-val jelölt szó helyes szófaji címkéje. A `t` ily módon úgy viselkedik, mintha ez lenne az a csúcs színe

Minden egyes $c = \{t_1, \ldots, t_k\}$ bizonytalansági osztályra és minden lehetséges t tagre $t \in c$ egy külön tanulási feladatot alakítottunk ki a színezett útvonal gráf háttértudással és $E^+(c,t)$, $E^-(c,t)$ pozitív és negatív példák halmazával:

- $E^+(c,t)$ az összes olyan atomot tartalmazza, `p(a)` atomot tartalmazza, a tréning adatbázisból, amelyben az `a` token a `c` bizonytalansági osztály egy előfordulása és a helyes címkéje `t`.
- $E^-(c,t)$ az összes olyan atomot tartalmazza, `p(a)` atomot tartalmazza, a tréning adatbázisból, amelyben az `a` token a `c` bizonytalansági osztály egy előfordulása és a helyes címkéje **nem** `t`.

Mivel az a probléma, hogy találjunk konzisztens hipotézist, amely `k` klózt tartalmaz, a legkisebb `k` számra egy NP teljes probléma [HorPh], ezért a *tabu-keresés* [Aart97] egy egyszerű változatát használtuk fel. A tabu méret 5 volt, a célfüggvény pedig maximalizálni a *relatív frekvencia* függvényt $n_{correct}/n$, ahol $n_{correct}$ a helyesen (a tréninggel megegyezően) minősített esetek száma, n pedig a hipotézissel lefedett tréning példák száma. A talált klózt akkor fogadta el az algoritmus, a pontossága a tréning adatokon legalább 90% volt, és a tréning példák legalább 10%-át lefedte. A *lefedni* ige azt jelenti itt, hogy a klóz alkalmazható egy adott példára és van valamilyen eredménye is.

4.7.8. A RIBL tanuló algoritmus alkalmazása

A RIBL egy példány alapú ILP tanulási módszer, amelyet először Emde és Wettschereck ismertetett [Emde96], később Bohnebeck, Horváth, és Wrobel továbbfejlesztette [BohnT]. Bár nagyszámú felhasználó által adható paraméter vezérli a RIBL-ben használt hasonlósági mértéket a fő probléma, hogy hogyan válasszuk meg az egyértelműsítési feladat megfelelő reprezentációját. Két különböző reprezentációt is közreadunk az alábbiakban.

---

[48] Probably Approximately Correct (valószűleg közelítőleg helyes). A tanulás egyik elméleti modellje.

A. Listával történő reprezentáció

Ebben a reprezentációban egy ternáris (három aritású) relációt használtunk a háttértudás reprezentációjára. Az első *input* argumentum a szó előfordulást azonosítja, a második és a harmadik argumentum az első argumentumban adott szóval szomszédos szavak címkéjét, a szövegkörnyezetet tartalmazza. Mind a két irányban csak legfeljebb két szót tekintettünk (illetve a címkéjüket). Minden egyes `c` = $\{t_1, \ldots, t_k\}$ bizonytalansági osztályra vettük mindazon `a` bizonytalan szavakat a tréning adatbázisból, amelyek a `c` osztályba tartoztak és egy `t(a)` tényt adtunk a héttér tudáshoz, ahol `t` az `a`-val azonosított szó helyes címkéje. Annak érdekében, hogy a két lista típusú argumentumot össze tudjuk hasonlítani, a lista *edit*-távolságot használtuk, amelyet a RIBL támogat [BohnT]. A lista abc-n (a lehetséges címkék halmazán) a triviális költségfüggvényt használtuk [BohnT].

B. Relációkkal történő reprezentáció

Ebben a reprezentációban egy bináris `R` relációval reprezentáljuk a szavak sorozatát olyan módon, hogy a háttértudás `R(a,b)` tényeket tartalmaz akkor és csak akkor, ha az `a` és `b` szavak követik egymást ebben a sorrendben. Minden egyes `a` tréning példára, amelynek a helyes annotációja `t`, egy `truetag(a,t)` tényt is hozzáadunk a háttértudáshoz.

Mint az előző reprezentációban itt is minden egyes `c` bizonytalansági osztályra külön tanulási feladatot tekintünk, ugyanazzal a háttértudással, ugyanúgy mint a listákkal történő reprezentáció esetén.

4.7.9. A kísérletekkel kapott első eredmények

Ebben a fejezetben összegezzük az öt tanuló algoritmussal kapott tapasztalati eredményeket. Mivel az AGLEARN külön extra háttértudást használt és szoros kapcsolatban áll a C 4.5-tel, ezért a C 4.5 és az AGLEARN eredményeket külön vetjük össze egymással.

A C4.5 algoritmussal kapott eredmények

A C4.5-tel kapott egyértelműsítő algoritmus a teszt adatokon csak 15 esetben nem tudott eredményt szolgáltatni, azaz a 2443 teszt eset 99,4%-ában adott valamilyen javaslatot a szófaji címkére. Az adott válaszok 1973 esetben voltak helyesek, ami 80,8%-a az összes (2443) teszt esetnek. Ez 81,3% relatív pontosságot jelent, ha csak azt a 2428 esetet számítjuk amikor a program eredményt adott.

Az AGLEARN algoritmussal kapott eredmények

A kibővített esetben, amikor az AGLEARN-t használtuk, az algoritmus a teszt adatokon 72 esetben nem tudott eredményt szolgáltatni, azaz a 2443 teszt eset 97,1%-ában adott valamilyen javaslatot a szófaji címkére. Az adott válaszok 2045 esetben voltak helyesek, ami 83,7%-a az összes (2443) teszt esetnek. Ez 86,3% relatív pontosságot jelent, ha csak azokat az 2428 esetet számítjuk amikor a program eredményt adott.

A Progol algoritmussal kapott eredmények

Ettől némileg eltérő eredményt kaptunk a Progol algoritmus alkalmazásánál: az algoritmus a teszt adatokon 837 esetben nem tudott eredményt szolgáltatni, azaz a 2443 teszt eset csupán 65,7%-ában adott valamilyen javaslatot a szófaji címkére. Megjegyezzük, hogy [Cuss97]-ben egy hasonló eredményt adtak közre. Másfelől, viszont a Progol 1482 esetben adott helyes választ, ami 60,7%-a az összes (2443) teszt esetnek. Ez kitűnő 92,3% relatív pontosságot jelent, ha csak azt az 1606 esetet számítjuk amikor a program eredményt adott.

A PHM algoritmussal kapott eredmények

A PHM módszerrel kapott algoritmus alkalmazásakor az algoritmus a teszt adatokon 478 esetben nem tudott eredményt szolgáltatni, azaz a 2443 teszt eset 80,4%-ában adott valamilyen javaslatot a szófaji címkére. Az algoritmus 1776 esetben adott helyes választ, ami 72,7%-a az összes (2443) teszt esetnek. Ez 90,4% relatív pontosságot jelent, ha csak azt az 1965 esetet számítjuk amikor a program eredményt adott.

A RIBL algoritmussal kapott eredmények

A listával történő reprezentáció esetén a szomszédok optimális száma 11 volt, amit az *egy-kimarad* módszerrel értékeltünk ki a tréning adatbázison. Mivel bizonyos esetekben, az ablakban szereplő szavak helyes címkéje nem ismert, erre egy speciális szimbólumot vezettünk be. Ennek a szimbólumnak a törlése, cseréje bármely más szimbólummal 1 költségű volt. Más rendszerekkel ellentétben a RIBL nem hagy eldöntetlen esetet. Az algoritmus 2072 esetben adott helyes választ, ami 84,8%-a az összes (2443) teszt esetnek. A legjobb tréning és (és teszt) pontosságot akkor értük el, amikor a mélység korlátot 3-ra állítottuk be a második reprezentáció esetén. Ekkor a RIBL helyes választ adott 1794 esetben, ami 73,4%-a az összes (2443) teszt esetnek. Ez az eredmény azt mutatta, hogy a pontosságban egy jelentős javulás érhető el, ha a lista edit-távolságot használjuk (azaz az első reprezentációt) az egyértelműsítési problémában (lásd még [BohnT]).

4.7.10. A kapott eredmények összefoglaló értékelése

A fejezet fennmaradó részében összefoglaljuk a fenti tapasztalati eredményeket. A RIBL esetében a listás (első) reprezentáció eredményét használtuk. A C 4.5, az AGLEARN és a RIBL nagyon jó döntési képességekkel rendelkezett, azaz csak kevés eldöntetlen esetet hagyott meg. A Progol és a PHM ellenben a teszt példák egy jelentős részében nem tudott dönteni. Másfelől viszont a Progol és a PHM rendelkezett a legjobb relatív pontossággal azokban az esetekben, amikor dönteni tudott. A Progol valamivel jobb eredményt ért el, mint a PHM. Megjegyezzük, hogy az extra nyelvészeti háttértudás, amit az AGLEARN-ben használtunk megnövelte az eldöntetlen esetek számát, és párhuzamosan a pontosságot is a C 4.5-höz képest.

Ahhoz, hogy az előző eredményeket összehasonlíthassuk, az eldöntetlen esetekben egy további, egyszerű lexikon gyakoriságokon alapuló egyértelműsítő algoritmust alkalmaztunk. Ennek az egyszerű módszernek önmagában alkalmazva a relatív pontossága 76,1% volt a 2443 teszt eseten. A végeredmény a 4.25. táblázatban látható. Ez a táblázat tartalmazza a szavankénti pontosságot is az írásjelek nélkül. A relatív pontosság sorban a 2443 teszt példán végrehajtott futtatás eredménye van a megvizsgált 27 bizonytalansági osztályon.

	C 4.5	AGLEARN	Progol	PHM	RIBL (listák)
Relatív pontosság (%)	81,0	84,8	82,8	84,6	84,8
Szavankénti pontosság (%)	97,60	98,03	97,80	98,00	98,03

4.25. Táblázat Az öt különböző tanuló algoritmussal kapott eredmény

4.7.11. Egyértelműsítő algoritmusok soros kapcsolása

Amint azt korábban említettük, a fő célunk nem az volt, hogy összehasonlítsuk az öt tanuló rendszert, hanem hogy kiválasszuk egy lehetséges működő kombinált egyértelműsítő rendszer komponenseit. Ebben az irányban tett első lépés az algoritmusok soros kapcsolásának vizsgálata volt. A kombinált rendszerekben először olyan komponenseket keresünk, amelyek nagyon pontosak, de a példák kis részére tudnak eredményt kiszámítani (Progol és/vagy PHM) majd haladunk visszafelé a RIBL felé, (amely minden esetben ad megoldást), de ez utóbbiakat csak azokra az esetekre alkalmazzuk, amit az előzőek eldöntetlenül hagytak. Az eredményeket a 4.26. Táblázat tartalmazza, a pontosságot a 2443 teszt esetre nézve számítottuk ki.

Bár a soros kapcsolás egy egyszerű kombináció szemben a [Halt98]-ban ismertetett más módszerekkel, a 4.26. Táblázat mégis jelzi a benne levő ígéretes lehetőséget. Érdekes megfigyelés, hogy a Progolnak a PHM-mel történő összekapcsolása nem növelte a pontosságot. Ez azt mutatja, hogy a PHM a megmaradt tokeneken már gyengének bizonyult.

	Progol RIBL	PHM RIBL	PHM RIBL
Relatív pontosság (%)	85,7	86,5	86,0

4.26. Táblázat Néhány sorosan kapcsolt algoritmus pontossága

4.7.12. A tanuló algoritmusokkal kapcsolatos tapasztalatok összegzése

Ebben a fejezetben összefoglaltuk öt kiválasztott ILP és nem-ILP tanuló algoritmus alkalmazásával kapcsolatos eredményeinket a magyar nyelvi szófaji egyértelműsítési problémára. A célja ennek a kísérletnek nem az volt, hogy összehasonlítsuk ezeket a rendszereket (hiszen teljesen különböző elvi alapokon működnek), hanem inkább az, hogy a magyar nyelv szokatlan és nehéz tulajdonságainak (nagy címke halmaz, szabad szórend) hatását megvizsgáljuk. Eredményeink azt mutatták, hogy a magyar nyelv ténylegesen kihívást jelent az egyértelműségi szabályok tanulásában, akár ILP, akár *propozícionális* tanuló algoritmusokat használunk.

Részletesebben, a problémának néhány általánosan megfigyelhető tulajdonsága valószínűleg átvihető más egyszerűbb nyelvekről a magyarra. Először is, a memória bázisú módszerek különösen alkalmasak egyértelműsítő algoritmusok tanulására [Halt98]. A 4.25. táblázat adatait ismerve ez lehet a mi esetünkben is a helyzet: az eset-alapú ILP tanulási módszer a RIBL nagyon jó eredményt mutatott ezen a doménen mind a fedésben mind pedig a pontosság tekintetében. Másodszor, amint azt Cussens [Cuss97] publikálta, hogy a nyelvészeti háttértudás nagy fontosságú és nem hagyható figyelmen kívül, amikor egy valóban használható módszert kívánunk létrehozni. Kísérleteinkben még a legegyszerűbb nyelvészeti háttértudás, amit az AGLEARN esetén a C 4.5 adatfájlhoz adtunk figyelemre méltó javulást eredményezett. Harmadszor, ahogy Cussens [Cuss97] kísérleteiben az angol nyelvre megmutatta, és mi a magyar nyelv esetén is ugyanezt észleltük: mindkét indukciós ILP módszer (Prolog és PHM) nagyon jó pontossági eredményeket ért el, azokban az esetekben, amikor dönteni tudtak. Azonban az eldöntetlen esetek száma viszonylag magas volt ezekben az esetekben, ami azt mutatja, hogy szükségük lenne még további általánosítási képességekre, amit például megfelelő háttértudás hozzáadásával lehetne biztosítani.

Folytatva a megkezdett munkát, a következőkben ezért a nyelvészeti háttértudás kialakítására teszünk erőfeszítéseket, amit nyelvész szakértők bevonásával tervezünk elérni, azért hogy megnövelhessük a pontosságot. A RIBL esetében speciálisan ez azt is jelenti, hogy egy szakértő által adott edit költség mátrix fogja helyettesíteni a jelen tanulmányban használt triviálist. A PHM esetén a fejlesztést a *lokális-keresés* használatában látjuk. Végül ismerve az algoritmusok kombinációjának eredményeit [Halt98] további bonyolultabb kombinációkat is ki fogunk próbálni.

5. Magyar nyelvű összefoglaló

A dolgozatban a szerző beszámol az elmúlt években elért tudományos eredményeiről. A szerző kutatási területe a gépi tanuló algoritmusokhoz és azok gyakorlati alkalmazásaihoz kapcsolódik. A téma a mesterséges intelligencia tárgykörébe tartozó kutatási ág. A tanuló algoritmusok a számítógépek olyan alkalmazását jelentik, amelyben a tanulás képességével rendelkező számítógépes rendszerek módosítani tudják saját működésüket a használat során nyert tapasztalatok alapján. Az általános célú tanuló algoritmusok egy kezdeti hipotézis valamint tréning példák felhasználásával javított és a példákkal konzisztens hipotéziseket képesek előállítani.

A tanuló algoritmusok alkalmazásának ugyancsak fontos területe a természetesnyelv-feldolgozás. Egy természetes nyelv milliós nagyságrendben tartalmaz szavakat. Ragozó (agglutináló) nyelvek esetén, mint amilyen a magyar is, a sok rag és jel miatt a különböző szóalakok száma elérheti a 10 milliót. Az írott szöveg megértéséhez a kulcs az összefüggések és a nyelvi szerkezetek elemzése, amelyhez a szavak morfo-szintaktikai attribútumait is felhasználja. Az Internet elterjedésével a mindenki által könnyen elérhető írott szöveges információ mennyisége drámai módon növekedett. Ennek a heterogén, soknyelvű anyagnak a bármilyen kis mértékű rendezése, feldolgozása, kivonatolása is nagy fontosságú, ezért került a természetesnyelv-feldolgozás a tudományos érdeklődés középpontjába. A természetesnyelv-feldolgozással szorosan kapcsolatban áll a beszédfelismerés. A tanuló algoritmusok hatékonyan alkalmazhatók például a beszédfelismerésben, a szófaji egyértelműsítésben, a főnévi szerkezetek kijelölésében. Az irodalom az alábbiak szerint határozza meg a tanuló programok fogalmát:

1.1. Definíció: Adottak a következők: Z egy számítógépes program, E tapasztalati tények (tréning adatok) egy halmaza, T elvégzendő (teszt) feladatok egy halmaza és P egy végrehajtási teljesítmény mérték. A Z program tanul az E gyakorlati tapasztalatokból, ha a Z programot az E gyakorlati példák feldolgozása után ismételten lefuttatva a T teszt feladatokon, a Z program P mérték szerinti teljesítménye javul.

A Z programot reprezentálhatjuk egy mesterséges neurális hálózattal, egy Markov hálózattal, illetve utasítások (kiszámítási szabályok, képletek vagy formulák) sorozatával. A tanulásnak ezt az utóbbi reprezentációt használó ágát általában a szimbolikus tanulás néven ismerik. A szimbolikus tanuláson belül különösen fontos terület a szabályhalmaz tanulás. Szabályokon *„ha-akkor"* típusú formulákat kell érteni, amelyekkel definiálni lehet egy új relációt ismert relációk felhasználásával. A szabályhalmazok tanulására szolgáló algoritmusokat nevezzük szabályalapú tanuló algoritmusoknak.

A tanuló programok maguk is *algoritmusok alapján működnek*, azaz a hipotéziseiket valamilyen szabályszerűség szerint *generálják* (egy keresési teret járnak be). A programok a rendelkezésükre álló idő alatt nagy keresési tereket tudnak megvizsgálni, és közben folyamatosan ellenőrzik a hipotéziseiket.

1.2. Definíció: Az induktív tanulási hipotézis: ha egy megtalált hipotézis jól közelíti a megtanulandó formulát (szabályt, predikátumot) egy elegendően nagy példa halmazon, akkor az jól fogja közelíteni a megtanulandó formulát a példák között elő nem fordult esetekben is.

Az induktív tanulási hipotézis teremti meg az alapot a hitelesített tanulási adatbázisok felhasználására. Számos gyakorlati esetben ugyanis a feladat megfogalmazásán túlmenően további elméleti keret nem áll rendelkezésre, az egyetlen forrás egy adatbázis, amelyben tapasztalati adatok találhatók a tanulmányozandó jelenségre vonatkozóan. A tanuló algoritmusok által adott eredmény minőségét az határozza meg leginkább, hogy a felhasznált tréning adatbázis mennyire elfogadott, szabványos és hiteles.

Az Európai Közösség két ízben is támogatott egy olyan kutatási projektet, amely témája az ILP (Inductive Logic Programming) a logikai programok (elsőrendű logikai formulák) tanulása volt. Az első a BRA (Basic Research Area) 6020 „ILP" ESPRIT projekt az elméleti alapok lerakására irányult 1993–1996 között. A második az LTR (Long Term Research) 20237 „ILP2" ESPRIT projekt az ILP tanuló algoritmusok lehetséges gyakorlati alkalmazási területeinek kijelölésére irányult 1996–1999 között. A Szegedi Tudományegyetem társintézmény volt mindkét projektben. A szerző tevékenyen részt vett a projektek kutatási-fejlesztési munkáiban.

A dolgozatban a szerző ismertet egy saját fejlesztésű interaktív tanuló algoritmust, a későbbiekben pedig ennek és más szabályalapú tanuló algoritmusoknak gyakorlati alkalmazása területén elért tudományos eredményeit. A 2. Fejezetben a szabályalapú tanuló algoritmusok néhány jelentősebb képviselője kerül bemutatásra. A 3. és a 4. Fejezet tartalmazza a szerző által elért tudományos eredményeket.

A 3. Fejezetben az IMPUT abduktív tanuló algoritmus ismertetésére kerül sor, amely a szerző saját eredménye. Az IMPUT algoritmust a szerző tervezte és implementálta, majd számos alkalmazás kifejlesztésére is sor került. Az IMPUT rendszer sikeresen alkalmazható volt a beszédfelismerésben magyar magánhangzók felismerésére, illetve EKG hullámok szintaxisának tanulására.

Az IMPUT rendszer a korábban kifejlesztett SPECTRE algoritmus egy jelentős továbbfejlesztése. A tanuló algoritmus a beépített IDTS hibakereső modulnak köszönhetően jelentősen javította a megtanult program minőségét. Bár a megtanult programok egyformán helyesek voltak mindkét tanuló program esetében, az IMPUT-tal kapott eredmények azonban sokkal tömörebbek és könnyebben érthetők voltak a SPECTRE eredményeivel összehasonlítva. A hibakereső rendszer használata megköveteli egy tanár jelenlétét.[49] A tanuló algoritmus hatékonyan fel tudja használni a tanár által a tanulás közben pótlólagosan bevitt információkat a hipotézis előállításához.

Az IMPUT rendszer IDTS hibakereső modulját a szerző és munkatársai közösen készítették az „ILP" BRA 6020 ESPRIT projektben. Segítségével az aktuális hipotézist és a tanulási példákat felhasználva lokalizálható a hiba helye, ami miatt az aktuális hipotézis nem viselkedik megfelelően. Az aktuális hipotézist az adott helyen megváltoztatva egy újabb hipotézis kapható. A transzformáció sorozat bizonyos feltételek esetén bizonyíthatóan véget ér, sőt el is jut egy, a példákkal konzisztens hipotézishez. Léteznek azonban olyan tanulási példák, amelyek esetében az algoritmus nem tud eljutni a megfelelő hipotézishez.

A 4. Fejezetben több a természetesnyelv-feldolgozáshoz kapcsolódó alkalmazás kerül bemutatásra. A szerző vezette be a természetes nyelvi egyértelműsítési feladat megoldásában a bizonytalansági-osztály fogalmát. Saját eredmény a bizonytalansági osztályokra alapozott tanulási modell kidolgozása több tanuló algoritmusra (C 4.5, Progol és AGLEARN). A szerző dolgozta ki az IMPUT algoritmus alkalmazási modelljét a magyar magánhangzók felismerésére. A THALES természetes nyelvi interfész attribútum nyelvtan alapú objektum memóriájának kifejlesztése is a szerző saját munkája.

A beszédfelismerésben a tanuló algoritmusok feladata könnyen definiálható: adott néhány hang minta, amelynek a szöveg megfeleltetése ismert. A feladat egy ismeretlen digitalizált hullám szöveges megfelelőjének előállítása. Az első igen fontos feladat a folyamatosan érkező hullámok szegmentálása, azaz elemi alkotórészekre (fonémákra) bontása. A következőkben az egyes fonémák azonosítása a cél, amely feladatra a különböző tanuló algoritmusok alkalmazása magától értetődik. Statisztikus módszereket már régóta használnak beszédfelismerésre. A magyar magánhangzók akusztikus szempontból viszonylag egyszerűek, néhány domináns spektrális összetevő szuperpozíciójának tekinthetők. Természetesen a frekvencia és intenzitás értékek a beszélőtől függnek, továbbá ezek az értékek időben is változhatnak. A szerző egy tanulási modellt dolgozott ki, amely az IMPUT tanuló algoritmus alkalmazására épült. A rendszer szabály-halmazokat tanult meg öt magyar magánhangzó azonosítására. További magánhangzók felismerésére a rendszer alkalmassá tehető, mássalhangzók esetében azonban további akusztikus tulajdonságok beépítése szükséges.

A természetes nyelvekkel kapcsolatos kutatások egyik ága a természetes nyelvi interfészek. Az ilyen típusú programok informatikai előképzettség nélküli természetes nyelven történő kommunikációt biztosítanak a felhasználó és a számítógépek között. A természetes nyelvi interfészek nem engedik meg a teljes természetes nyelv használatát, annak csak egy jól definiált részhalmaza használható. A mondatok alapvetően olyan parancsok kiadására alkalmasak, amelyek az adott alkalmazási terület objektumaival kapcsolatosan bizonyos tevékenységek elvégzésére utasítják a számítógépet. A gyakorlatban is jól működő rendszer egyik kulcsa a pontosan definiált parancs szintaxis és az erre épülő szemantika. A szerző és munkatársai egy attribútum nyelvtan alapú, nagymértékben általánosítható interfész generátor rendszert készítettek, amelynek egy alkalmazása volt a THALES síkgeometriai szerkesztések végrehajtására alkalmas felhasználói interfész. A szerző saját eredménye az objektumokat tároló memória, a szimbólumtábla attribútum nyelvtanos specifikáción alapuló előállítása volt, továbbá a kezeléshez szükséges eljárások forráskódjának generálása.

---

[49] A szakirodalomban a tanárt sokszor nevezik *bölcsnek* (oracle). A dolgozatban is ez az elnevezést fordul elő.

A természetes nyelvek feldolgozása közben több egymástól jól elkülöníthető feladatot definiáltak. A feldolgozás a szöveg strukturális felbontásával kezdődik, az adatállományokban található szövegeket fejezetekre, bekezdésekre, mondatokra és szavakra bontják fel. Fontos feladat a szófaji elemzés, amelynek során a szavakhoz hozzárendelik a lehetséges morfo-szintaktikai attribútumaikat. A természetes nyelvekben gyakran fordulnak elő többjelentésű szavak. A többjelentésű szavak címkéi közül a szövegkörnyezetnek megfelelő címke kiválasztását nevezik szófaji egyértelműsítésnek. Az egyértelműsítési feladat megoldására már régóta alkalmaznak statisztikai (HMM) tanuló algoritmusokat. Az utóbbi időben több szabályalapú megközelítést is publikáltak. A tanuló algoritmusok alkalmazásának egyik problémája a tanulási példák nagy száma. A szerző kidolgozott egy a gyakorlatban is hatékonyan működő módszert a tanulási példák számának csökkentésére a feladat kisebb, egymástól függetlenül megoldható tanulási feladatokra bontásával. A kisebb feladatok megoldásaként kapott szabályhalmazok egyesítésével kapható meg a végeredmény. A szerző által kidolgozott módszer segítségével több különböző tanuló algoritmus felhasználásával határoztak meg egyértelműsítési szabályokat, amelyeket egyértelműsítő programokba építettek be. A szerző által kidolgozott elvek alapján készültek el a C 4.5, a Progol és az AGLEARN algoritmusok alkalmazásai. Végül a szerző munkatársaival egy összehasonlító tanulmányt készített öt különböző tanuló algoritmusra alkalmazására vonatkozóan, továbbá sikeres kísérleteket végeztek a különböző szabályrendszerek kombinálására.

A dolgozat elkészítéséhez felhasznál publikációk megjelenését követően a szerző az IKTA 27/2000 projekt egyik irányítójaként részt vett egy magyar nyelvű 1 millió szavas tanuló adatbázis fejlesztésében. Az adatbázis jelentős emberi erőforrások bevonásával 2002-ben készült el. A szerző és munkatársai kísérleteket kezdtek tanuló algoritmusok futtatására és egyértelműsítési szabályok tanulására. A megtanult szabályok segítségével gyakorlatban is működő egyértelműsítő rendszert készítettek. Az NKFP 02/17/2001 projektben a névszói szerkezetek struktúráját vizsgálják, amelynek során a főnévi szerkezet nyelvtanának tanulására az IMPUT rendszert alkalmazzák.

6. Summary in English

Rule-based Learning Methods and their Applications to Natural Language Processing

The present thesis summarizes the scientific results obtained by the author over the past couple of years. His main research area is learning algorithms and their practical applications. This topic is a branch of artificial intelligence. Learning algorithms are advanced applications of computer systems, in which programs, capable of learning, can modify their own behavior upon experience. A general purpose learning algorithm can infer modified hypotheses consistent with the learning examples starting from an initial hypothesis and experience.

An important application area of learning is natural language processing. A natural language contains around 1 million words. In the case of agglutinating languages, such as Hungarian, the number of different forms of words can reach up to10 million due to the large number of stems. The key to understanding written text is analyzing the language structures and phrases utilizing the morpho-syntactic attributes of words. Since the Internet has become so popular all over the world, the amount of publicly available textual information increased dramatically. Any small advance in sorting, processing, and summarizing this heterogeneous multilingual information has gained great importance. This is why natural language processing has become the center of scientific interest. Also, speech recognition is strongly connected to natural language processing. Learning algorithms have been effectively applied in speech recognition, part-of-speech tagging and NP-chunking. A concept of *learning* is described as follows according to the literature:

1.1. Definition [Mit97]**:** A computer program Z is said to learn from set of experiences E with respect to some set of (test) tasks T, and performance measure P, if its performance at test tasks T as measured by P, improves after processing the experiences E.

Program Z can be represented by artificial neural-networks, Markov models, or series of statements (computational rules, formulae). Determining hypotheses in this latter form is known as symbolic learning. Learning rule-sets plays an important role within symbolic learning. Rules are "*if-then*" rules, by which new relations can be defined as deducting them to other known relations. Algorithms learning rule-sets are called rule-based learning algorithms. Learning programs themselves are also working upon a built-in algorithm, that means their hypotheses are generated according to some rules (traversing a search space). Programs within a given amount of time can walk through large search spaces while checking their hypotheses.

1.2. Definition: The inductive learning hypothesis [Mit97]: any hypothesis found to approximate the target function (formula, predicate) well over a sufficiently large set of training examples will also approximate the target function over other unobserved examples.

The inductive learning hypothesis makes a basis of using certified learning databases. In many practical cases there is no sufficient theoretical framework, the only thing at one's disposal is a database that contains observations relating to the studied phenomenon. The quality of the results obtained by learning algorithms is determined mainly in what measure the learning database is approved, standardized and certified.

The European Community supported two research and development projects on ILP (Inductive Logic Programming), learning logic programs (first order logical formulae). The first project, BRA (Basic Research Area) 6020 "ILP" ESPRIT, focused on founding the theoretical grounds between 1993-1996. The second project, LTR (Long Term Research) 20237 "ILP2", aimed at collecting possible practical application fields between the years 1996-1999. University of Szeged was partner in both of these projects. The author actively took part in the R&D works.

In his thesis, the author presents an interactive inductive learning algorithm, followed by the description of his scientific results concerning practical applications of rule-based learning methods. In chapter 2, several notable examples of learning algorithm are presented. Chapter 3 and 4 contain the scientific achievements of the author.

In chapter 3, the IMPUT abductive learning tool – one of the author's achievement – is presented. It has been designed and implemented by the author, and several applications have been developed for it. IMPUT could be applicable in speech-recognition for recognizing Hungarian vowels and for learning syntax-rules of ECG waveforms.

IMPUT is a significant improvement of the original SPECTRE algorithm. Owing to the built-in IDTS debugging module, IMPUT could significantly improve the quality of the learnt program. Although, both programs worked equally well on examples, the resulting programs obtained by IMPUT were more compact and understandable, and have less clauses than that obtained by SPECTRE. The IDTS debugging system requires the presence of a tutor[50]. The learning algorithm could efficiently exploit the additional information entered by the *oracle* during the learning process while making its next hypothesis.

The IDTS debugging subsystem was made by the author and his colleagues in the frame of the "ILP" BRA 6020 ESPRIT project. By its help, the exact location of the error causing the wrong behavior of the actual hypothesis can be determined given the actual hypothesis and the learning examples. Correcting the actual hypothesis at the determined place, found by the IDTS, a new hypothesis can be obtained. This series of transformation terminates if certain conditions are held, moreover, it converges to a hypothesis consistent with the learning examples. On the other hand, there may exist some learning tasks, in cases of which IMPUT cannot reach a consistent hypothesis.

Chapter 4 deals with applications to natural language processing. The author introduced the concept of ambiguity-classes in part-of-speech tagging tasks. He elaborated learning models based on ambiguity-classes for several learning algorithms (C 4.5, Progol, and AGLEARN). The author developed an application model of the IMPUT algorithm for recognizing Hungarian vowels. The author developed an object-memory (symbol table) based on attribute grammar specification for THALES natural language interface.

The role of learning algorithms in speech-recognition can be easily defined given some sound samples and its textual representation. The task of a learning algorithm is then to determine a program, which can map an unknown digitized sound to text. The first very important step is segmenting the incoming continuous wave, that is decomposing it to atomic elements (*phonemes*). The goal of the following steps is to identify each phoneme, for which the application of learning algorithms is straightforward. Concerning its acoustics, Hungarian vowels are simple; they can be considered superposition of some dominant spectral components. Naturally, the intensity and frequency values depend on the speaker and they also vary in time. Statistical methods for phoneme recognition have been used for many years. The author developed a learning model for the IMPUT interactive learning algorithm for identifying five Hungarian vowels. The framework can be extended to recognize more vowels, but in case of consonants it requires new acoustic features to be added.

A branch of natural language research deals with natural language interfaces. Such interfaces ensure a channel between the user and the computer, by which the exchange of information is done in one's mother tongue therefore it does not require any skill in information technology. Natural language interfaces only allow a subset of the language to be used. The sentences are suitable for issuing commands to do a specific action with objects of an application area. The key to a system that works well in practice is the precise definition of syntax, and the definition of semantics based on syntax. The author and his colleagues developed a generator system based on attribute grammars for producing natural language interfaces. It has been applied to generate the THALES prototype, a natural language interface for making plane geometry constructions. The author developed attribute grammar specification formalism for the object-memory (symbol table) of that natural language interface and the environment for generating compilable source code from the specification.

In natural language processing many specific, well separable tasks have been defined. The processing begins with the structural decomposition of texts; the data is split to chapters, divisions, paragraphs, sentences and finally words. The next step is the morphological parsing, during which each token is labeled by its morpho-syntactic attributes. Ambiguous words occur in natural languages quite often. The disambiguation process that selects the label, which is correct in the textual context, is called part-of-speech tagging. For solving the disambiguation task, statistical methods (like Hidden Markov Models) have been used for many years. Several rule-based approaches were published recently. In applications of learning algorithms, the large number of examples causes difficulty. The author introduced a method for decreasing the number of learning examples by

---

[50] This person is often called *oracle* in the literature, so this will be used throughout the thesis.

dividing the learning tasks into numerous smaller ones. The method proved to work well in practice. The rule-set for the whole learning task can be obtained by uniting the rule-sets of the small learning tasks. Applying the above method, disambiguation rules for part-of-speech tagging of Hungarian language have been determined, using different learning tools. The disambiguation rules have been built into part-of-speech tagger programs. The application models for C 4.5, Progol, and AGLEARN were developed on the basis of the author's work. The author and his colleagues wrote a comparative study on using five different learning algorithms for learning disambiguation rules for part-of-speech tagging. In each learning task, the same learning data set was used and successful experiments were done to combine different rule-sets.

After the appearance of the papers included in this thesis, the author, as member of the steering committee of the IKTA 27/2000 project, led the development of the Hungarian disambiguated corpus containing 1 million words. The development of the database was finished in 2002 with much human effort. The author and his colleagues started experiments running learning tools and determining disambiguation rules. The determined rule-sets have been built into a disambiguation system. In the project NKFP 2/017/2001 they study the syntactic structure of Hungarian noun-phrases. The IMPUT system is applied for refining the initial noun-phrase syntax rules.

Irodalom

[Aart97] Aarts, E. H. L. and Lenstra, J. K. (editors): Local Search in Combinatorial Optimization, Wiley-Interscience Publishing, Chichester, England, in series Discrete Mathematics and Optimization, (1997).

[Alb91] Alblas, H.: Introduction to Attribute Grammars. LNCS **545**, Springer Verlag, 1–16. (1991).

[Ade94] Adé, H., Malfait, B., and De Raedt, L.: RUTH: an ILP Theory Revision System, in Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems (ISMIS94), (1994).

[Ade95] Adé, H. and Boström H.: JIGSAW: puzzling together RUTH and SPECTRE, in Proceedings of the 8th European Conference on Machine Learning, Springer Verlag, (1995).

[AleA90] Alexin, Z., Dombi, J., Fábricz, K., Gyimóthy, T. and Horváth, T.: CONSTRUCTOR: A Natural Language Interface Based on Attribute Grammars, in Acta Cybernetica Vol **9** No **3** pp. 247–255 Szeged, Hungary (1990).

[Ale90] Alexin, Z., Gyimóthy, T. and Horváth T.: Attribute Grammar Specification for a Natural Language Understanding Interface in Proceedings of the First International Workshop on Attribute Grammar Applications (WAGA90) Paris, France, September 19–21, in the LNCS series Vol **461** pp. 313–326 Springer Verlag (1990).

[Ale95] Alexin, Z., Gyimóthy, T. and Boström, H.: Integrating Algorithmic Debugging and Unfolding Transformation in an Interactive Learner, in *Proceedings of the 5th International Workshop on Inductive Logic Programming* (ILP95) Leuven, Belgie, Eds: Luc de Raedt, pp. 437–452 (1995).

[Ale96] Alexin, Z., Gyimóthy, T. and Boström, H.: Integrating Algorithmic Debugging and Unfolding Transformation in an Interactive Learner, in *Proceedings of the 12th European Conference on Artificial Intelligence* (ECAI96), Budapest, Hungary, Eds: Wolfgang Wahlster, pp. 403–407, John Wiley & Son's Ltd. (1996).

[AleI97] Alexin, Z., Gyimóthy, T. and Boström, H.: IMPUT: An Interactive Learning Tool based on Program Specialization, Intelligent Data Analysis (IDA) Journal, Vol **1** No **4**, Elsevier Holland (1997).

[Ale97] Alexin, Z., Csirik, J., Gyimóthy, T., Jelasity, M. and Tóth, L.: Learning Phonetic Rules in a Speech Recognition System in *Proceedings of the 7th International Workshop on Inductive Logic Programming* (ILP97) Prague, Czech Republic, in the LNAI series Vol **1297** pp. 37–44 Springer Verlag (1997).

[AleW99] Alexin, Z., Zvada, Sz., Gyimóthy, T.: Application of AGLEARN for Hungarian Part-of-speech Tagging in *Proceedings of the second Workshop on Attribute Grammar Applications* (WAGA99) Amsterdam, The Netherlands 26 March 133–152, INRIA Rocquencourt (1999).

[Bain92] Bain, M. and Muggleton, S.: Non-monotonic Learning, in *Inductive Logic Programming* (editor: Muggleton, S.), pp. 145–161, Academic Press, London, (1992).

[Berg88] Bergadano, F. and Giordana, A.: A Knowledge Intensive Approach to Concept Induction, in *Proceedings of the 5th International Conference on Machine Learning* (ICML88), pp. 305–317, Morgan Kaufmann, CA, (1988).

[Bha99] Bhavani Thuraisingham: Data Mining, Technologies, Techniques, Tools and Trends, CRC Press (1999) ISBN: 0-8493-1815-7.

[Bis95] Bishop, C.: Neural networks for pattern recognition, Oxford University Press, http://neural-server.aston.ac.uk/NNPR/index.html, (1995).

[Biter95] Biter, N. N. and Espy-Wilson, C. Y.: A Signal Representation of Speech Based on Phonetic Features, in Proceedings of 5th Annual Dual-Use Techniques and Applications Conference, pp. 310–315, (1995).

[Bloc99] Blockeel, H., Džeroski, S. and Grbovič, J.: Using TILDE to analyse biological and chemical data on river water quality, Jožef Stefan Institute, Technical Report May 1999.

[BohnT] Bohnebeck, U., Horváth, T., and Wrobel, S.: Term Comparisons in First-Order Similarity Measures, in *Proceedings of the 8th International Conference on Inductive Logic Programming* (ILP98), Madison, Wisconsin, USA, in LNAI series, Vol **1446**, (Eds.: David Page), pp. 65–79, Springer Verlag, (1998).

[Bohn98] Bohnebeck, U., Sälter, W., Horváth, T., Wrobel, S. and Blohm, D.: Measuring Similarity of RNA Structures by Relational Instance-Based Learning: A First Step toward Detecting RNA Signal Structures in Silico, in *Proceedings of German Conference on Bioinformatics*, Cologne (1998).

[Bol95] Bolla, K.: Magyar fonetikai atlasz, A szegmentálás hangszerkezet elemei, Nemzeti Tankönyvkiadó Rt. Budapest, in Hungarian, (1995).

[Bos94] Boström, H. and Idestam-Almquist, P.: Specialization of Logic Programs by Pruning SLD-trees, in *Proceedings of the 4th International Workshop on Inductive Logic Programming* (ILP94) Bad-Honnef, Bonn, Germany, pp. 31–47 (1994).

[Bos95] Boström, H.: Specialization of Recursive Predicates, in *Proceedings of the 5th International Workshop on Inductive Logic Programming* (ILP95) Leuven, Belgie, Eds: Luc de Raedt, pp. 92–106 (1995).

[Bos99] Boström, H., and Asker, L.: Combining Divide-and-Conquer and Separate-and-Conquer for Efficient and Effective Rule Induction, in *Proceedings in the 9th Workshop on Inductive Logic Programming (ILP99)*, in the LNAI series Vol **1634** pp. 33–43, Springer Verlag (1999).

[Boye93] Boye, J., Paakki, J. and Maluszýnski, j.: Synthesis of Directional Information for Functional Logic Programs, in *Proceedings of the 3rd International Workshop on Static Analysis*, Padova, Italy, in LNCS series Vol **724**, pp. 165–177, Springer Verlag (1993).

[Chen96] Nienhuys-Cheng, S. H. and de Wolf, R.: A Complete Method for Program Specialization Based on Unfolding, in *Proceedings of the 12th European Conference on Artificial Intelligence* (ECAI96), Budapest, Hungary, pp. 438–442, John Wiley & Sons, (1996).

[Chis79] Chistowitch, L. A. and Lublinskaya, V. V.: The Center of Gravity Effect in Vowel Spectra and Critical Distance between Formants, in *Hearing Research*, 185–195, (1979).

[Cla89] Clark, P. and Niblett, R.: The CN2 induction algorithm, in Machine Learning, Vol **3**, 261–284, (1989).

[Coh91] Cohen, W. W.: The Generality and Overgenerality, in Proceedings of the 8th International Workshop on Machine Learning, pp. 490–494, Morgan Kaufmann, (1991).

[Coh92] Cohen, W. W.: Compiling Prior Knowledge Into Explicit Bias, in *Proceedings of the 9th International Workshop on Machine Learning* (ICML92), pp. 102–110, Morgan Kaufmann, (1992).

[Cuss97] Cussens, J.: Part-of-Speech Tagging Using Progol in *Proceedings of the 7th International Workshop on Inductive Logic Programming* (ILP97) Prague, Czech Republic, in the LNAI series Vol **1297** pp. 93–108 Springer Verlag (1997).

[Cuss99] Cussens, J., Džeroski, S. and Erjaveč, T.: Morphosyntactic tagging of Slovene using Progol, in *Proceedings in the 9th Workshop on Inductive Logic Programming (ILP99)*, in the LNAI series Vol **1634** pp. 68–79, Springer Verlag (1999) http://www.cs.bris.ac.uk/~ilp99/.

[Cut92] Cutting, D., Kupiec, J., Pederson, J., and Sibun, P.: A Practical Part-of-speech Tagger, in *Proceedings of the 3rd Conference on Applied Natural Language Processing* (ANLP92), pp. 133–140, ACL, (1992).

[Dael98] Daelemans, W., van den Bosch, A., and Zavrel, J.: Rapid Development of NLP Modules with Memory-Based Learning, in *Proceedings of ELSNET* in Wonderland, Utrecht, pp. 105–113 (1998).

[DeRa92] De Raedt, L. and Bruynooghe, M.: Belief Updating from Integrity Constraints and Queries, in *Artificial Intelligence*, **53 2–3**, pp 291–307, (1992).

[Dej00] Déjean, H.: Learning Syntactic Structures with XML, in *Proceedings of the 4th Computational Natural Language Learning Workshop* (CoNLL-2000), Lisbon, Portugal, pp. 133–135, (2000).

Der85] Deransart, P. and Maluszyński, J.: Relating Logic Programs and Attribute Grammars, in *Journal of Logic Programming* Vol **2**, pp. 119–156, (1985).

[Der88] Deransart, P., Jourdan, M., and Lorho, B.: Attribute Grammars – Definitions, Systems and Bibliography. LNCS **323**, Springer Verlag. (1988).

[Der93] Deransart, P. and Maluszyński, J.: A Grammatical View of Logic Programming, The MIT Press. (1993).

[Dol92] Dolšak, B. and Muggleton, S.: The application of inductive logic programming to finite element mesh design, in *Inductive Logic Programming* (Muggleton, S.: editor), pp. 453–472, Academic Press, London, (1992).

[Dzer92] Džeroski, S. and Dolšak, B.: Comparison of ilp systems on the problem of finite element mesh design, in *Proceedings of the 6th ISSEK Workshop*, Jožef Stefan Institute, Ljubljana, Slovenia, (1992).

[Dzer93] Džeroski, S. and Lavrač, N.: Inductive Learning in Deductive Databases, in *IEEE Transactions on Knowledge and Data Engineering*, Vol. **5**. No. 6. pp. 939–949 (1993).

[Dzer98] Džeroski, S., Jacobs, N., Molina, M. And Moure, C.: ILP experiments in detecting traffic problems, in *Proceedings of 10th European Conference on Machine Learning*, 61–66, Springer Verlag (1998).

[Dzer99] Džeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B. And Van Laer, W.: Experiments in predicting biodegradability, in *Proceedings in the 9th Workshop on Inductive Logic Programming (ILP99)*, LNAI series Vol **1634** pp. 80–91, Springer Verlag (1999) http://www.cs.bris.ac.uk/~ilp99/.

[Eine98] Eineborg, M. and Lindberg, N.: Induction of Constraint Grammar-Rules Using Progol, in *Proceedings of the 8th International Conference on Inductive Logic Programming* (ILP98), Madison, Wisconsin, USA, in LNAI series, Vol **1446**, pp. 116–124, (1998).

[Eine99] Lindberg, N. and Eineborg, M.: Improving Part-of-speech Disambiguation Rules by Adding Linguistic Knowledge, in *Proceedings of 9th International Workshop on Inductive Logic Programming* (ILP99) Bled, Slovenia, in the LNAI series Vol **1634** pp. 186–197, Springer Verlag (1999). http://www.cs.bris.ac.uk/~ilp99/

[Emde96] Emde W. Wettschereck, D.: Relational Instance Based Learning, in Machine Learning, in *Proceedings 13th International Conference on Machine Learning*, (Eds.: Lorenza Saitta), pp. 122–130, Morgan Kaufmann Publishers (1996). ftp://ftp.gmd.de/ml-archive/GMD/papers/ML79.ps

[Eri96] Erika Van Baelen and Luc De Raedt: Analysis and Prediction of Piano Performances Using Inductive Logic Programming, in *Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP96)*, in the LNAI series Vol **1314** pp. 55–71, Springer Verlag (1996).

[Erja97] Erjavec, T. and Monachini, M.: Specification and Notation for Lexicon Encoding, Copernicus project 106 "MULTEXT-EAST", Work Package WP1 – Task 1.1 Deliverable D1.1F, (1997).

[Fay92] Fayyad, U. M. and Irani, K. B.: On the handling of continuous-valued attributes in decision tree generation, in Machine Learning, **8**, pp. 87–102, (1992).

[Gin93] Ginsberg, M.: Essentials of Artificial Intelligence, ISBN: 1-55860-221-6, Morgan Kaufmann, San Mateo, (1993).

[Gyim88] Gyimóthy, T., Horváth, T., Kocsis, F. and Toczki, J.: Incremental Algorithms in PROF-LP in *Proceedings of the Workshop on Compiler Compiler and High Speed Compilation* (CCHSC88), Berlin GDR, in LNCS series Vol **371** pp. 93–103 Springer Verlag (1988).

[Gyim97] Gyimóthy, T., Horváth, T.: Learning Semantic Functions of Attribute Grammars, in *Nordic Journal of Computing*, Vol **4**, pp. 287–302 (1997).
http://www.cs.helsinki.fi/njc/References/gyimothyh1997:287.html

[Halt98] van Halteren, H., Zavrel, J., and Daelemans, W.: Improving Data Driven Wordclass Tagging by System Combination, in *Proceedings of COLING-ACL'98*, Montreal, Canada, pp. 491–497, (1998).

[Hay94] Haykin, S.: Neural networks: a comprehensive foundation, MacMillen College Publishing Company, http://www.prenhall.com/~ray/002/u29977/u29977.html, (1994).

[Hed89] Hedin, G.: An Object-oriented Notation for Attribute Grammars, in *Proceedings of the European Conference on Object Oriented Programming* (ECOOP'89), Nottingham.

[Hor93] Horváth, T., Gyimóthy, T., Alexin, Z. and Kocsis, F.: Interactive Diagnosis and Testing Logic Programs, in *Proceedings of the 3rd Symposium on Programming Languages and Software Tools* (SPLST93), Kääriku, Estonia, pp. 34–46, (1993).

[Hor96] Horváth, T. and Turán, Gy.: Learning Logic Programs with Structured Background Knowledge, *Advances in Inductive Logic Programming*, (Eds.: L. De Raedt), pp. 172–191, IOS Press (1996).

[Hor97] Horváth, T., Sloan, R. H., and Turán Gy.: Learning Logic Programs by Using the Product Homomorphism Method, in *Proceedings of the 10th Annual Conference on COmputational Learning Theory* (COLT97), pp. 10–20, ACM Press, New York, (1997).

[HorPh] Horváth, T.: Learning logic programs with structured background knowledge, German National Research Center for Information Technology, PhD. Thesis, (1999).

[Hor99] Horváth, T., Alexin, Z., Gyimóthy, T., Wrobel, S.: Application of Different Learning Methods to Hungarian Part-of-speech Tagging, in *Proceedings of 9th International Workshop on Inductive Logic Programming* (ILP99) Bled, Slovenia, in the LNAI series Vol **1634** pp. 128–139, Springer Verlag (1999) http://www.cs.bris.ac.uk/~ilp99/

[Hueb94] Huebner, K. And Carson-Berndsen, J.: Phoneme Recognition Using Acoustic Events, Verbmobil Technical Report No. 15, June (1994).

[Jaco01] Nico Jacobs and Hendrik Blockeel: From Shell Logs to Shell Scripts, in *Proceedings of the 11th International Workshop on Inductive Logic Programming (ILP2001)*, in the LNAI series Vol **2157** pp. 80–90, Springer Verlag (2001).

[Jel95] Jelasity, M. and Dombi, J.: GAS, an Approach on Modeling Species in Genetic Algorithms, in *Proceedings of EA'95*, (1995).

[Joh00] Johansson, C.: A Context Sensitive Maximum Likelihood Approach to Chunking, in *Proceedings of the 4th Computational Natural Language Learning Workshop* (CoNLL-2000), Lisbon, Portugal, pp. 136–138, (2000).

[Jeli97] Jelinek, F.: Statistical Methods for Speech Recognition, Cambridge, MA:MIT Press, (1997).

[Kas80] Kastens, U.: Ordered Attribute Grammars, *Acta Informatica*, Vol **13** pp. 229–256 (1980).

[Kaz98] Kazakov, D. and Manandhar, S.: A hybrid approach to word segmentation, in *Proceedings of the 8th International Conference on Inductive Logic Programming* (ILP98), Madison, Wisconsin, USA, in LNAI series, Vol **1446**, pp. 125–134, (1998).

[Kij92] Kijsirikul, B., Numao, M and Shimura, M: Discrimination-based constructive induction of logic programs, in *Proceedings of the 10th International Conference on Artificial Intelligence*, pp. 44–49, San Jose, California, USA, (1992).

[King96] King, R., Muggleton, S., Srinivasan, A. and Sternberg, M.: Structure-activity relationships derived by machine learning: the use of atoms and their bond connectives to predict mutagenecity by inductive logic programming, in *Proceedings of the National Academy of Sciences*, 93:438–442, (1996).

[Knu68] Knuth, D. E.: Semantics of Context-Free Languages in *Mathematical Systems Theory* **2**, 2, pp. 127–145. (1968), Correction: *Mathematical Systems Theory* **5**, 1, pp. 95–96 (1971).

[Kók94] Kókai, G., Alexin, Z., Kocsis, F.: The IDTS System and its Application for Learning Logic Programs, in *Proceedings of the 6th International Conference on Artificial Intelligence and Information Control Systems of Robots* (AIICSR-94), Smolenice Castle, Slovakia, pp. 315–320, (1994).

[Kók96] Kókai, G., Alexin, Z., and Gyimóthy, T.: Analyzing and Learning ECG Waveforms, in *Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP96)*, in the LNAI series Vol **1314** pp. 127–145, (1996).

[KókI96] Kókai, G., Alexin, Z. And Gyimóthy, T.: Learning Biomedical Patterns, in *Proceedings of the 9th Exhibition and Symposium on Industrial Applications of Prolog* (INAP96), Tokyo, Japan, pp. 159–168 (1996).

[KókP96] Kókai, G., Alexin, Z., and Gyimóthy, T.: Classifying ECG Waveforms in Prolog, in *Proceedings of the 4th International Conference of Practical Application of Prolog* (PAP96), London, United Kingdom, pp. 193–221, (1996).

[Kók97] Kókai, G., Alexin, Z., and Gyimóthy, T.: Application of Inductive Logic Programming for Learning ECG Waveforms, in Proceedings of the 6th Conference on Artificial Intelligence in Medicine, Grenoble, France, poster, in LNAI series **1211**, pp. 126–129, Springer Verlag (1997).

[Kos87] Koskimies, K. and Paakki, J.: TOOLS – a Unifying Approach to Object-oriented Language Interpretation, in *Proceedings of ACM Sigplan'87*, Sigplan Notices, Vol **22** No **7**, (1987).

[Lam93] Lamel, L. F.: A knowledge-based system for stop consonant identification based on speech spectogram reading, Computer Speech and Language, pp. 169–191, (1993).

[Lav94] Lavrač, N. and Džeroski, S.: Inductive Logic Programming Techniques and Applications, Ellis Horwood Ltd. ISBN: 0-13-457870-8, (1994).

[Ling91] Ling, C. X.: Non-monotonic Specialization, in *Proceedings of the 1st International Workshop on Inductive Logic Programming* (ILP91), Portugal, pp. 59–68, (1991).

[Lloy93] Lloyd, John Wylie: Foundations of Logic Programming, Second, extended edition, ISBN: 3-540-18199-7 Springer Verlag (1993).

[Mam95] Mammachandran, R. P., Mammone, R. J. (editors): Modern methods of Speech Processing, Kluwer Academic, (1995).

[Man98] Manandhar, S., Džeroski, S. and Erjaveč, T.: Learning multilingual morphology with CLOG, in *Proceedings of the 8th International Conference on Inductive Logic Programming* (ILP98), Madison, Wisconsin, USA, in LNAI series, Vol **1446**, pp. 135–144, Springer Verlag (1998).

[Man99] Mani, I. and Maybury, M. T.: Advances in Automatic Summarization, ISBN: 0-262-13359-8, MIT Press, Cambridge MA, (1999).

[Man00] Manning, D. C., Schütze, H.: Foundations of Statistical Natural Language Processing, The MIT Press, third printing, ISBN: 0-262-13360-1, (2000).

[Mart83] Martin, P., Appelt, Pereira, F.: Transportability and Generality in a Natural Language Interface System, in *Proceedings of IJCAI'83*, **1**, 573–581, (1983).

[Megy98] Megyesi Beáta: Brill's Rule Based Part-of-Speech Tagger for Hungarian, University of Stockholm, (1998), master thesis

[Mer94] Merialdo, B.: Tagging English Text with a Probabilistic Model, in *Computational Linguistics*, **20**, (2), pp. 155–171, (1994).

[Mit97] Tom M. Mitchel: Machine Learning, The McGrew-Hill Companies Co. (1997) ISBN: 0-07-115467-1.

[Mizo96] Fumio Mizoguchi, Hayato Ohwada, Makiko Daidjo and Shiroteru Shirato: Learning Rules that Classify Ocular Fundus Images for Glaucoma Diagnosis, in *Proceedings of the 6th International Workshop on Inductive Logic Programming* (ILP96), in the LNAI series Vol **1314** pp. 146–159, (1996).

[Mich80] Michalski, R. S.: Pattern Recognition as Rule-Guided Inductive Inference, in *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol **2**, pp. 349–361, (1980).

[Moo91] Mooney, R. J. and Ourston, D.: Constructive Induction in Theory Refinement, in *Proceedings of the 8th International Workshop on Machine Learning* (ICML91), pp. 178–182, Morgan Kaufmann, (1991).

[Moo96] Mooney, R. J.: Inductive Logic Programming for Natural Language Processing, in *Proceedings of the 6th International Workshop on Inductive Logic Programming* (ILP96), in the LNAI series Vol **1314** pp. 3–22, (1996).

[Moze98] Mozetič, I.: Secondary Structure Prediction by Inductive Logic Programming, in *Proceedings of the 3rd Meeting on the Critical Assessment of Techniques for Protein Structure Precdiction*, CASP3, pp A-26, Asilomar, CA, (1998).

[Mugg88] Muggleton, S. and Buntime, W.: Machine invention of first-order predicates by inverting resolution, in *Proceedings of the 5th International Conference on Machine Learning*, Ann Arbor, Michigan, USA, pp. 339–352, (1988).

[Mugg90] Muggleton, S, and Feng, C.: Efficient induction of logic programs, in *Proceedings of the 1st Conference on Algorithmic Learning Theory*, Ohmsha Publishing, Tokio (1990).

[Mugg92] Muggleton, S., King, R. and Sternberg, M.: Protein secondary structure prediction using logic-based machine learning, in *Protein Engineering*, Vol **5**, No **7**, 647–657, (1992).

[MuggI] Muggleton, S.: Inductive Logic Programming, Academic Press, London. (1992).

[Mugg94] Muggleton, S. and De Raedt, L.: Inductive Logic Programming: Theory and Methods, in *Journal of Logic Programming*, Vol **12** (1994).

[Mugg95] Muggleton, S.: Inverse Entailment and Progol, in *New Generation Computing, Special issue on Inductive Logic Programming* Vol **13**, No **3–4**, pp. 245–286 Ohmsha Publishing (1995).

[MuggT] Muggleton, S., Bryant, C. H., Srinivasan, A., Gloger I. S., Lawrence, M., Whittaker, A., Topp, S. and Rawlings, C. J.: Are grammatical representations useful for learning from biological sequence data? – a case study, University of York, Technical Report

[Ner97] Nerode, A. and Shore, R. A.: Logic for Applications, second edition, ISBN: 0-387-94893-7, Springer Verlag (1997).

[Ora98] Oravecz, Cs.: Part-of-Speech Tagging in the Hungarian National Corpus – a Case Study, *techincal report*, Research Institute for Linguistics at Hungarian Academy of Sciences (1998).

[Ost88] Ostrand, T. J. And Balker, M. J.: The Category-Partition Method for Specifying and Generating Functional Tests, in *CACM* **31:6** June, pp. 676–686, (1988).

[Our90] Ourston, D. and Mooney, R. J.: Changing the Rules: A Comprehensive Approach to Theory Refinement, in *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 815–820, MIT Press, (1990).

[Paa90] Paakki, J.: A Logic-Based Modification of Attribute Grammars for Practical Compiler Writing, in *Proceedings of the 7th International Conference on Logic Programming* (Eds.: Warren, D. H. D., Szeredi, P.), Jerusalem, The MIT Press, pp. 203–217 (1990).

[Paa94] Paakki, J., Gyimóthy, T., and Horváth, T.: Effective Algorithmic Debugging for Interactive Logic Programming, in *Proceedings of the 4th International Workshop on Inductive Logic Programming* (ILP94), Bad-Honnef, Bonn, Germany, pp. 175–194, (1994).

[Paz91] Pazzani, M., Brunk, C., and Silverstein, G.: A Knowledge-Intensive Approach to Learning Relational Concepts, in *Proceedings of the 8th International Workshop on Machine Learning* (ICML91), pp. 432–436, Morgan Kaufmann, (1991).

[Paz93] Pazzani, M. and Brunk, C.: Finding Accurate Frontiers: A Knowledge-Intensive Approach to Relational Learning, in Proceedings of the 11th National Conference on Artificial Intelligence, pp. 328–334, Morgan Kaufmann, (1993).

[Per80] Pereira, F. C. N., Warren, D. H. D.: Definite Clause Grammars for Language Analysis – A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence* **13**, pp. 231–278 (1980).

[Plot70] Plotkin, G. D.: A note on inductive generalization, in *Machine Intelligence* (Meltzer, B. and Michie, D. editors), Vol **5**, New York, Elsevier North-Holland (1970).

[Plot71] Plotkin, G. D.: Automatic Methods of Inductive Inference, *PhD thesis*, Edinburgh University, August, (1971).

[PTE99] Srinivasan, A., King, R. D., and Bristol, D. W.: An Assessment of ILP-Assisted Models for Toxicology and the PTE-3 Experiment, in *Proceedings of 9th International Workshop on Inductive Logic Programming* (ILP99) Bled, Slovenia, in the LNAI series Vol **1634** pp. 291–302, Springer Verlag (1999) http://www.cs.bris.ac.uk/~ilp99/.

[Quin86] Quinlan, J. R.: Induction of Decision Trees, *Machine Learning*, Vol **1** No **1** pp. 81–106.

[Quin90] Quinlan, J. R.: Learning logical definitions from relations, in Machine Learning, Vol **5**, pp. 239–266, (1990).

[Quin93] Quinlan, J. R.: C 4.5: Programs for Machine Learning, Morgan Kaufmann Publisher (1993).

[Quin01] René Quiniou, Marie-Odile Cordier, Guy Cerrault and Feng Wang: Application of ILP to Cardiac Arrythmia Characterization for Chronicle Recognition, in *Proceedings of the 11th International Workshop on Inductive Logic Programming (ILP2001)*, in the LNAI series Vol **2157** pp. 220–227, (2001).

[Rab78] Rabiner, L. R. and Schafer, R. W.: Digital Processing of Speech Signals, Prentice Hall (1978).

[Rab93] Rabiner, L. R. and Juang, B. H.: Fundamentals of Speech Recognition, Prentice Hall (1993).

[Rich91] Richards, B. L. and Mooney, R. J.: First-Order Theory Revision, in Proceedings of the 8th International Workshop on Machine Learning (ICML91), pp. 447-451, Morgan Kaufmann, (1991).

[Rob98] Roberts, S., Van Laer, W., Jacobs, N., Muggleton, S. and Broughton, J.: A comparison of ILP and propositional systems on propositional data, in *Proceedings of 8th International Workshop on Inductive Logic Programming (ILP98),* Medison, Wisconsin, USA, in the LNAI series Vol **1446** pp. 291–299, Springer Verlag (1998).

[Schr88] Schröder, M.: Evaluating User Utterances in Natural Language Interfaces to Databases, in *Computers and Artificial Intelligence*, Vol 7 No 4 pp. 317–337.

[Seb02] Sebastiani, F.: Machine Learning in Automated Text Categorization, in ACM Computing Surveys, Vol **34** No **1**, pp. 1–47, March, (2002).

[Shap83] Shapiro, E. Y.: Algorithmic Program Debugging, MIT Press, (1983).

[Srin99] Srinivasan, A. and King, R. D.: Using Inductive Logic Programming to Construct Structure-Activity Relationships, in *Proceedings of the AAAI Spring Symposium on Predictive Toxicology,* AAAI Press, Menlo Park, CA, (1999)

[Skor84] Skordalakis, E. and Papakonstantinou G.: Towards an Attribute Grammar for the Description of ECG Waveforms, in *Proceedings of the 7th International Conference on Pattern Recognition* (1984).

[Skor90] Skordalakis, E.: ECG Analysis, in *Syntactic and Structural Pattern Recognition Theory and Applications* (eds: Bunke, H. and Sanfeliu, A.), World Scientific, 499–533, (1990).

[Tam84] Tamaki, H. and Sato, T.: Unfold/Fold transformation of Logic Programs, in *Proceedings of the 2nd International Logic Programming Conference*, Uppsala, Sweden, pp. 127–138, (1984).

[Toc88] Toczki, J., Gyimóthy, T., Horváth, T., and Kocsis, F.: Generation of modular compilers in PROF-LP in *Proceedings of the Workshop on Compiler Compiler and High Speed Compilation* (CCHSC88), Berlin GDR, in LNCS series Vol **371** pp. 104–111 Springer Verlag (1988).

[TELRI] TELRI: East meets West — A Compendium of Multilingual Resources Eds: Tomaž Erjavec, Lawson, A. and Romary, L. http://www.ids-mannheim.de/telri/cdrom.html

[Turc98] Turcotte, M., Muggleton, S. and Sternberg, M.: Application of inductive logic programming to discover rules governing the three-dimensional topology of protein structures, in *Proceedings of the 8th International Conference on Inductive Logic Programming* (ILP98), in the LNAI series Vol **1446** pp. 53–64. Springer Verlag (1998).

[Wai90] Waibel, A. and Lee, K. (editors): Readings in Speech Recognition, Morgan Kaufmann, (1990).

[Wog91] Wogulis, J.: Revising Relational Domain Theories, in *Proceedings of the 8th International Workshop on Machine Learning* (ICML91), pp. 462–466, Morgan Kaufmann, (1991).

[Wrob93] Wrobel, S.: On the Proper Definition of Minimality on Specialization and Theory Revision, in Proceedings of the European Conference on Machine Learning, pp. 65–82, Springer Verlag, (1993).

[Zel94] Zelle, J. M., Mooney, R. J. and Konvisser, J. B.: Combining Top-down and Bottom-up Techniques in Inductive Logic Programming, in *Proceedings of the 11th International Conference on Machine Learning*, pp. 343–351, Morgan Kaufmann, (1994).

[Zhou00] Zhou, GouDong, Su, Jian, and Tey, TongGuan: Hybrid Text Chunking, in *Proceedings of the 4th Computational Natural Language Learning Workshop* (CoNLL-2000), Lisbon, Portugal, pp. 163–165, (2000).

[Zue85] Zue, V. W.: The Use of Speech Knowledge in Automatic Speech Recognition, in *Proceedings of IEEE*, Vol. **73**, No. **11**, pp. 1602–1615, (1985).